

# NP-Compleitude

Análise de Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

1 Introdução

2  $\mathcal{P}$  vs  $\mathcal{NP}$



# Sumário

---

## 1 Introdução



# Introdução

---

- Até agora vimos diversos problemas que podem ser resolvidos em tempo polinomial.
- Todos os problemas podem ser resolvidos em tempo polinomial?



# Introdução

---

- Existem problemas que podem ser resolvidos em tempo polinomial:
  - ▶ Ordenação.
  - ▶ Compressão LZ77.
  - ▶ Seleção de Eventos.
  - ▶ Subsequência comum mais longa.



# Introdução

---

- Existem problemas em que, independente do tempo, é impossível resolvê-los com os modelos computacionais relevantes (tese de Church-Turing):
  - ▶ Problema da parada.
  - ▶ Problema da Correspondência de Post.



# Introdução

---

- Existem problemas em que não são conhecidas soluções eficientes para eles:
  - ▶ Ciclo Hamiltoniano.
  - ▶ Caixeiro Viajante.
  - ▶ Cobertura de Vértices.
  - ▶ SAT.



# Complexidade Computacional

---

- O campo de Complexidade Computacional estuda a dificuldade dos problemas.
- Procura classificar os problemas em classes de complexidade.
- Por que precisamos estudar Complexidade Computacional?





# Complexidade Computacional

---

- O estudo da Complexidade Computacional permite:
  - ▶ Verificar a dificuldade de um problema.
  - ▶ Estabelecer a dificuldade de um problema baseado em outro.
  - ▶ Procurar por soluções aproximadas ou heurísticas, caso o problema seja identificado como “difícil”.



# Problemas Tratáveis

---

- O que é um problema difícil?
- O é uma solução eficiente?
- Qual o conceito formal de eficiência?
- Quando um problema é considerável tratável?



# Problemas Tratáveis

---

- Na literatura, problemas tratáveis admitem uma solução em  $O(n^k)$  para algum  $k > 0$ .
- Soluções polinomiais são ditas “eficientes”, mesmo que o polinômio tenha grau alto.
- Por exemplo:  $O(n^{10})$ .
- Isto não corresponde totalmente à prática, mas necessitamos deste parâmetro de eficiência para desenvolver a teoria.



# Problemas Tratáveis

---

- De maneira geral, problemas que admitem solução polinomial são ditos tratáveis.
- Problemas que não admitem solução em tempo polinomial são denominados de intratáveis pela literatura.



## A Classe $\mathcal{P}$

---

- A classe de complexidade  $\mathcal{P}$  corresponde aos problemas de decisão que podem ser resolvidos em tempo  $O(n^k)$  por algum algoritmo.



## Exemplos de Problemas Tratáveis

---

- SORT: Ordenação.
  - ▶ Entrada:  $V = (v_0, v_1, \dots, v_{n-1})$ .
  - ▶ Saída:  $V' = (v'_0, v'_1, \dots, v'_{n-1})$ , uma permutação de  $V$ , tal que  $v'_i < v'_{i+1}$  para  $0 \leq i < n - 1$ .
- Pode ser resolvido em tempo  $O(n \lg n)$ .



## Exemplos de Problemas Tratáveis

---

- TOP-K: Seleção dos  $k$  maiores.
  - ▶ Entrada:  $V = (v_0, v_1, \dots, v_{n-1})$  e um inteiro  $k$ .
  - ▶ Saída:  $V' = (v'_0, v'_1, \dots, v'_{k-1})$ , os  $k$  maiores elementos de  $V$ .
- Pode ser resolvido em tempo  $O(n \lg k)$ .



# Exemplos de Problemas Tratáveis

---

- HUFF: Codificação Huffman.
  - ▶ Entrada:  $T$  sobre um alfabeto  $\Sigma$ .
  - ▶ Saída:  $T'$  codificado de acordo com o código de Huffman.
- Pode ser resolvido em tempo  $O(n \lg |\Sigma|)$ .





## Exemplos de Problemas Tratáveis

---

- LCS: Subsequência Comum mais Longa.
  - ▶ Entrada: *strings*  $X[0, n - 1]$  e  $Y[0, m - 1]$ .
  - ▶ Saída:  $Z$ , uma subsequência comum de  $X$  e  $Y$  de tamanho maximal.
- Pode ser resolvido em tempo  $O(n \cdot m) \subseteq O(\max\{n, m\}^2)$ .



## Exemplos de Problemas Tratáveis

---

- EVENT: Seleção de eventos.
  - ▶ Entrada: tempo de eventos  $E = ([l_0, r_0], [l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ .
  - ▶ Saída: tamanho do conjunto maximal de eventos compatíveis.
- Pode ser resolvido em tempo  $O(n \lg n)$ .



# Problemas de Decisão

---

## Definição (Problemas de Decisão)

Problemas de decisão são aqueles que para qualquer instância  $I$  do problema a saída será **Verdadeiro** ou **Falso**.

- Podemos converter os problemas visto anteriormente em sua versão de decisão.
- A nossa teoria será baseada em problemas de decisão, que não são mais difíceis que os problemas de otimização ou busca.



## A Classe $\mathcal{P}$

---

- A classe de complexidade  $\mathcal{P}$  corresponde aos problemas de **decisão** que podem ser resolvidos em tempo  $O(n^k)$  por algum algoritmo.



## A Classe $\mathcal{P}$

---

- Vários Problemas interessantes encontram-se em  $\mathcal{P}$ .
- Podem ser resolvidos em tempo eficiente.
- Nem todos os problemas possuem esta propriedade. . .
- Estudaremos agora problemas decidíveis, que possuem solução, mas não conhecemos uma solução **eficiente** para eles.



# A Classe $\mathcal{NP}$

---

- A classe  $\mathcal{NP}$  engloba problemas **decisão** em que conseguem ser verificados em tempo polinomial.
- Um problema pode ser verificado se, dado um “Certificado” da solução, é possível verificar se ele está correto em um tempo polinomial.



## Exemplo de Problema em $\mathcal{NP}$

---

### SUBSET-SUM

- **Entrada:** Um conjunto de inteiros positivos  $S = s_0, s_1, \dots, s_{n-1}$  e um inteiro  $k$ .
- **Saída:**

**Verdadeiro,** se existe  $S' \subseteq S$  com  $\sum_{x \in S'} x = k$

**Falso,** caso contrário



## Exemplo de Problema em $\mathcal{NP}$

---

$$S = \{1, 3, 8, 13, 22, 37, 62, 47, 83, 20, 33, 100, 65\}$$

$$k = 215$$

- A saída para esta entrada é **Verdadeiro** ou **Falso**?





## Exemplo de Problema em $\mathcal{NP}$

---

$$S = \{1, 3, 8, 13, 22, 37, 62, 47, 83, 20, 33, 100, 65\}$$

$$k = 215$$

- A saída para esta entrada é **Verdadeiro** ou **Falso**?
- **Verdadeiro:**  $S' = \{8, 62, 47, 33, 65\}$



# SUBSET-SUM

---

---

**Algorithm 1:** SUBSET-SUM-SOLVER( $S, k$ )

---

**Input:**  $S, k$

**Output:** **True** se e somente se existe  $S' \subseteq S, \sum_{x \in S'} x = k$

1 **return** SUBSET-SUM-SOLVER( $S, k, 0, 0$ )

---



## SUBSET-SUM

---

---

**Algorithm 2:** SUBSET-SUM-SOLVER( $S, k, i, psum$ )

---

**Input:**  $S, k, i, psum$

**Output:** **True** se e somente se existe  $S' \subseteq S, \sum_{x \in S'} x = k$

```
1 if(  $psum = k$  )  
2   | return True  
3 else if(  $(psum > k) \vee (i == |S|)$  )  
4   | return False  
5  $b \leftarrow$  SUBSET-SUM-SOLVER( $S, k, i + 1, psum + S[i]$ )  
6  $b \leftarrow b \vee$  SUBSET-SUM-SOLVER( $S, k, i + 1, psum$ )  
7 return  $b$ 
```

---



## SUBSET-SUM

---

- Claramente o algoritmo anterior leva tempo  $\Omega(2^n)$ .
- Não conhecemos um algoritmo que leve tempo  $O(n^k)$ , para algum  $k > 0$ .
- No entanto conseguimos verificar o certificado em tempo polinomial:
  - ▶ Basta fazer a soma de  $\{8, 62, 47, 33, 65\}$ , no exemplo anterior.
- SUBSET-SUM  $\in \mathcal{NP}$ .

 $\mathcal{NP}$ 

---

## Definição (Classe $\mathcal{NP}$ )

Um problema está em  $\mathcal{NP}$  se existe um algoritmo verificador  $V$  para o problema que, dado uma instância  $I$  cuja saída é **Verdadeiro** e um certificado  $W$ ,  $V(I, W)$  diz **Verdadeiro** e roda em tempo polinomial.



# Sumário

---

## 2 $\mathcal{P}$ vs $\mathcal{NP}$



# $\mathcal{P}$ vs $\mathcal{NP}$

---

## Teorema ( $\mathcal{P} \subseteq \mathcal{NP}$ )

Todo problema que está em  $\mathcal{P}$  também está em  $\mathcal{NP}$ .

## Demonstração.

Todos os problemas em  $\mathcal{P}$  podem ser resolvidos em tempo polinomial, logo o verificador pode simplesmente ignorar o certificado e resolver o problema para obter a resposta.  $\square$



## $\mathcal{P}$ vs $\mathcal{NP}$

---

- Mostramos que  $\mathcal{P} \subseteq \mathcal{NP}$ .
- Será que todo problema que pode ser verificado em tempo polinomial também pode ser resolvido em tempo polinomial, isto é,  $\mathcal{NP} \subseteq \mathcal{P}$ ? Se este for o caso  $\mathcal{P} = \mathcal{NP}$ , caso contrário,  $\mathcal{P} \neq \mathcal{NP}$ .
- A resposta para esta pergunta vale 1 milhão de dólares: <http://www.claymath.org/millennium-problems/p-vs-np-problem>





# $\mathcal{P}$ vs $\mathcal{NP}$

---

- A chave para a resposta da questão  $\mathcal{P}$  vs  $\mathcal{NP}$  está nos problemas  $\mathcal{NP}$ -completos, pertencentes à  $\mathcal{NPC}$ , considerados os mais difíceis de  $\mathcal{NP}$ .
- Para compreender a classe  $\mathcal{NPC}$ , precisamos compreender o conceito de Redução de problemas, o qual veremos na próxima aula.



# $\mathcal{P}$ vs $\mathcal{NP}$

---

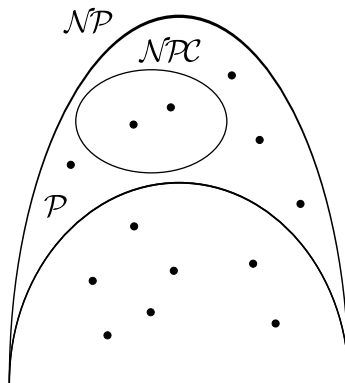


Figura: Conjecturando  $\mathcal{P} \neq \mathcal{NP}$



# $\mathcal{P}$ vs $\mathcal{NP}$

---

$$\mathcal{P} = \mathcal{NP} = \mathcal{NPC}$$

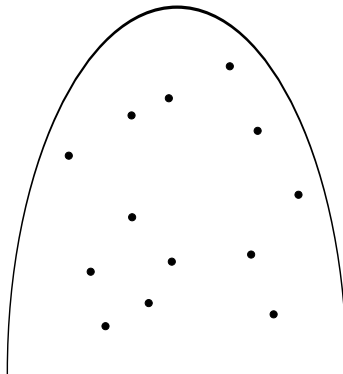


Figura: Conjecturando  $\mathcal{P} = \mathcal{NP}$