Revisão: P, NP e NP-Completude

Computabilidade e Complexidade

Márcio Nicolau

2025-11-10

Table of contents

List of Figures

Objetivos da Aula

- Solidificar a distinção fundamental entre resolver um problema (P) e verificar uma solução (NP).
- Reforçar o conceito de NP-Completude como a "fronteira da intratabilidade".
- Praticar a identificação de problemas em P e NP.
- Revisar a técnica de redução em tempo polinomial para provar NP-Completude.

Conteúdo

O Grande Mapa da Complexidade de Tempo

Deixamos para trás o mundo da decidibilidade e entramos no reino da eficiência. O foco agora não é se um problema *pode* ser resolvido, mas se ele pode ser resolvido em um tempo *razoável*.

Definições Essenciais (Revisão)

- **Classe P (Tempo Polinomial)**: Problemas que podem ser **resolvidos** eficientemente. Existe um algoritmo determinístico que os decide em tempo $O(n^k)$. Estes são os problemas "tratáveis".
- **Classe NP (Tempo Polinomial Não-Determinístico)**: Problemas para os quais uma solução proposta (um "certificado") pode ser **verificada** eficientemente.
- **NP-Completo (NPC)**: Um subconjunto de NP que contém os problemas "mais difíceis" da classe. Se qualquer problema NPC puder ser resolvido em tempo polinomial, então **todos** os problemas em NP também poderão, o que implicaria que P = NP.
- **Redutibilidade Polinomial ($A \leq_p B$)**: A ferramenta para provar que um problema B é *pelo menos tão difícil quanto* um problema A, de forma eficiente.

A relação mais importante a se lembrar é que $\mathbf{P} \subseteq \mathbf{NP}$. A grande questão em aberto é se essa inclusão é estrita ($\mathbf{P} \subset \mathbf{NP}$) ou não ($\mathbf{P} = \mathbf{NP}$).

Diagrama: O Mundo de NP (se P $\neq$ NP)

Resolver vs. Verificar: A Distinção Crucial

A intuição por trás de P vs. NP reside na diferença entre criar e criticar. É mais fácil criticar uma obra de arte (verificar) do que criar uma obra-prima do zero (resolver).

Característica	Problemas em P	Problemas em NP
Foco	**Resolver**	**Verificar**
Tempo	Resolução rápida (polinomial)	Verificação rápida (polinomial)
Exemplo	Encontrar o caminho mais curto em um mapa.	Dado um caminho, verificar se ele é o mais curto.
Status	Considerado "fácil" ou tratável.	Pode ser "difícil", mas as soluções são fáceis de reconhece

A Receita para Provar NP-Completude

Provar que um problema `X` é NP-Completo é um procedimento padrão com dois passos:

1. **Mostrar que $X \in \mathbf{NP}$**:
   - Defina qual é o "certificado" (a prova ou solução).
   - Descreva um algoritmo verificador que, dado o problema e o certificado, confirma a solução em tempo polinomial.
2. **Mostrar que X é NP-Difícil**:
   - Escolha um problema Y que você **já sabe** ser NP-Completo (geralmente 3-SAT).
   - Construa uma redução em tempo polinomial $Y \leq_p X$.
   - A redução é uma função f que transforma instâncias de Y em instâncias de X.
   - Prove que a transformação preserva a resposta: $w \in Y \iff f(w) \in X$.

Se ambos os passos forem bem-sucedidos, você provou que `X` é um dos "problemas mais difíceis em NP".

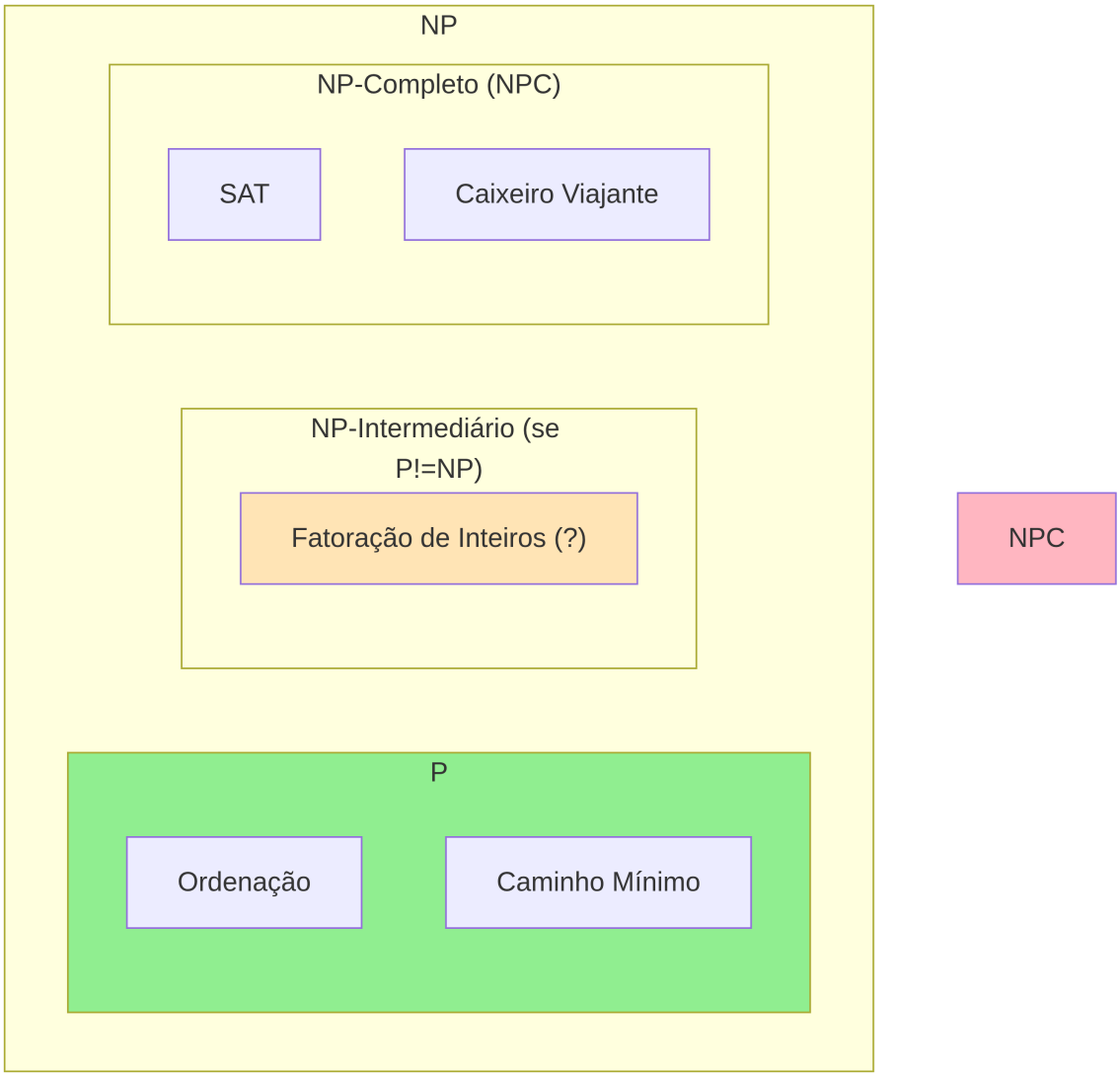

Figure 1: A estrutura acreditada das classes de complexidade, mostrando P, NP e os problemas NP-Completos.

Estudo de Caso: 3-SAT é NP-Completo

O problema **3-SAT** é uma versão restrita do SAT e é a ferramenta de redução mais comum.

Problema 3-SAT: Dada uma fórmula booleana em Forma Normal Conjuntiva (um `E` de cláusulas), onde cada cláusula é um `OU` de **exatamente três** literais (ex: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$), a fórmula é satisfatível?

Prova de que 3-SAT é NP-Completo: 1. **3-SAT $\in$ NP**: O certificado é uma atribuição de Verdadeiro/Falso para as variáveis. O verificador simplesmente substitui esses valores na fórmula e calcula o resultado. Isso é muito rápido (linear no tamanho da fórmula). 2. **3-SAT é NP-Difícil**: A prova é feita mostrando que SAT $\leq_p$ 3-SAT. Existe um procedimento engenhoso (e polinomial) que converte qualquer cláusula do SAT em um conjunto de cláusulas de 3 literais, preservando a satisfatibilidade.

Como 3-SAT é NP-Completo, ele se torna nosso novo "ponto de partida" para provar que outros problemas, especialmente em grafos, também são NP-Completos.

Implicações Práticas em Código

Quando confrontado com um problema que se suspeita ser NP-Completo na vida real, como otimizar a rota de entrega de pacotes (uma variante do TSP), a teoria nos diz o que fazer.

```
# Problema: Encontrar o menor tour no Caixeiro Viajante (TSP)
distancias = {
    ('A', 'B'): 10, ('A', 'C'): 15, ('A', 'D'): 20,
    ('B', 'A'): 10, ('B', 'C'): 35, ('B', 'D'): 25,
    ('C', 'A'): 15, ('C', 'B'): 35, ('C', 'D'): 30,
    ('D', 'A'): 20, ('D', 'B'): 25, ('D', 'C'): 30,
}
cidades = ['A', 'B', 'C', 'D']

def calcular_custo_tour(tour):
    custo = 0
    for i in range(len(tour) - 1):
        custo += distancias.get((tour[i], tour[i+1]), float('inf'))
    return custo

# Abordagem 1: Força Bruta (Exponencial, intratável para n > 15)
def tsp_forca_bruta(cidades):
    # Gera todas as permutações de cidades e encontra a de menor custo
    # Complexidade O(n!)
    # ... código omitido por ser muito lento ...
    print("Abordagem 1: Força Bruta (intratável para muitas cidades).")
    return "Um tour ótimo, mas demorou uma eternidade."

# Abordagem 2: Heurística do Vizinho Mais Próximo (Rápida, mas não garante o ótimo)
def tsp_heuristica_vizinho_proximo(cidades, inicio):
    tour = [inicio]
```

```
    nao_visitadas = set(cidades)
    nao_visitadas.remove(inicio)

    atual = inicio
    while nao_visitadas:
        proximo = min(nao_visitadas, key=lambda cidade: distancias.get((atual, cidade), float('inf')))
        tour.append(proximo)
        nao_visitadas.remove(proximo)
        atual = proximo

    tour.append(inicio) # Volta para o começo
    print("Abordagem 2: Heurística (rápida, solução 'boa o suficiente').")
    return tour

# Conclusão prática: usamos a heurística
tour_aproximado = tsp_heuristica_vizinho_proximo(cidades, 'A')
custo_aproximado = calcular_custo_tour(tour_aproximado)
print(f"Tour encontrado: {tour_aproximado} com custo {custo_aproximado}")
```

```
Abordagem 2: Heurística (rápida, solução 'boa o suficiente').
Tour encontrado: ['A', 'B', 'D', 'C', 'A'] com custo 80
```

Exercícios de Verificação

Atividade Prática de Revisão

1. **P, NP ou NPC?**: Classifique os seguintes problemas e justifique:
   - L_1: Dado um grafo, ele contém um triângulo (um clique de tamanho 3)?
   - L_2: Dado um inteiro N, ele possui um fator primo menor que k?
   - L_3: Dado um conjunto de tarefas com durações e prazos, é possível agendá-las em uma única máquina para cumprir todos os prazos? (Dica: Pense na dificuldade de encontrar a ordem certa).
2. **A Lógica da Redução**: Você está tentando provar que o problema `COLORAÇÃO DE GRAFOS` (colorir um grafo com k cores sem que vizinhos tenham a mesma cor) é NP-Completo. Você já sabe que ele está em NP. Qual das seguintes reduções seria o passo correto?
   a) `COLORAÇÃO` $\leq_p$ `3-SAT`
   b) `3-SAT` $\leq_p$ `COLORAÇÃO`
   c) `CAMINHO MÍNIMO` $\leq_p$ `COLORAÇÃO`
3. **A Consequência do P=NP**: Se fosse provado que P=NP, o que isso significaria para o conceito de "criatividade"? Argumente brevemente se isso eliminaria a necessidade de "insights" criativos para resolver problemas difíceis.

Referências Bibliográficas