



Resumo PAA

Sophia Carrazza

Análise Assintótica: <https://www.youtube.com/watch?v=pwn4y78MZFw> ✓

Relações de Recorrência: <https://www.youtube.com/watch?v=yBML5usRsy4> ✓

P vs NP: <https://www.youtube.com/watch?v=pQsdygaYcE4> ✓

NP e NP-Completo: <https://www.youtube.com/watch?v=VJucgdXGHyE> ✓

P, NP, NP-Completo e NP-Difícil: https://www.youtube.com/watch?v=ApRmVUOOY_o ✓

SAT e 3-SAT: <https://www.youtube.com/watch?v=GCw07nZckps> ✓

Set Cover e Vertex-Cover ✓

Relação de \leq_p (~~é~~ difícil quanto, etc) ✓

Tipos de quantificadores + frequentes ✓

Prova da definição de Big-O ✓

$X \in \text{NP-Completo}$:

- i) $\in \text{NP}$
 - ii) $\exists Y \in \text{NP-C} \mid Y \leq_p X$
- exist* ↗
- movendo que podemos reduzir Y para X* ↗
porque ai, provamos que o X é tão difícil quanto Y. Portanto, ele também é NP-Completo

Análise Assintótica

theta(Θ):

mostra como o tempo de execução cresce a medida que aumenta a entrada. (não é o tempo de relojão)

ex: algoritmo A($n \log n$)

* é tudo sobre a relevância de um termo no crescimento da função

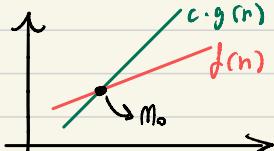
(ex: constantes não são tão relevantes perto de n^2)

$$\boxed{A} < \boxed{A}$$
$$c_1 + \dots + c_2 + \dots$$

big-O (O):

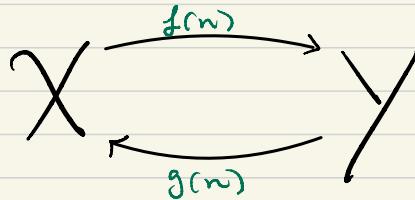
$f(n) = O(g(n))$ SSTE: $\exists c > 0, m_0 \geq 0 \mid f(n) \leq c \cdot g(n), \forall n \geq m_0$

é o limite superior!



redução polinomial

Se um problema X pode ser reduzível em tempo polinomial p Y, isso significa que qualquer instância X pode ser transformada em uma instância do problema Y dentro de um tempo polinomial.



- X se reduz para Y
- X é tão fácil quanto Y
- Y é tão difícil quanto X

* Se $X \leq_p Y$ e $Y \leq X$, logo: $X \equiv_p Y$ → todos os problemas NP-completos são equivalentes

NP-completos
equivalentes

Classes de Problemas

P: Polynomial Time

para problemas de decisão que podem ser resolvidos em tempo polynomial $\rightarrow O(n^2), O(m \log m), O(m^{203})$

NP: Nondeterministic Polynomial time

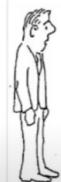
- para problemas de decisão em que todo resultado "sim" possui solução de tamanho polynomial e pode ser verificado em tempo polynomial.
- ou seja, é um problema em que não temos um algoritmo eficiente para解决-lo, mas tendo uma resposta em mão podemos checar-la em tempo polynomial.
- também é um problema que pode ser resolvido em tempo polynomial utilizando uma máquina de turing não-determinística.

*pode ser que o problema esteja em P se não encontrarmos uma resposta p
Ele, ou pode ser que não esteja em P.

NP-Completo:

- para um problema A que está contido em NP e, todo outro problema B em NP se reduz polynomialmente a A.
- são os problemas mais difíceis de NP.
- se conseguirmos resolver um problema NP-completo eficientemente (em tempo polynomial) saberemos que $P = NP$

Ele está em NP!



"I can't find an efficient algorithm, I guess I'm just too dumb."

Ele é NP-completo!



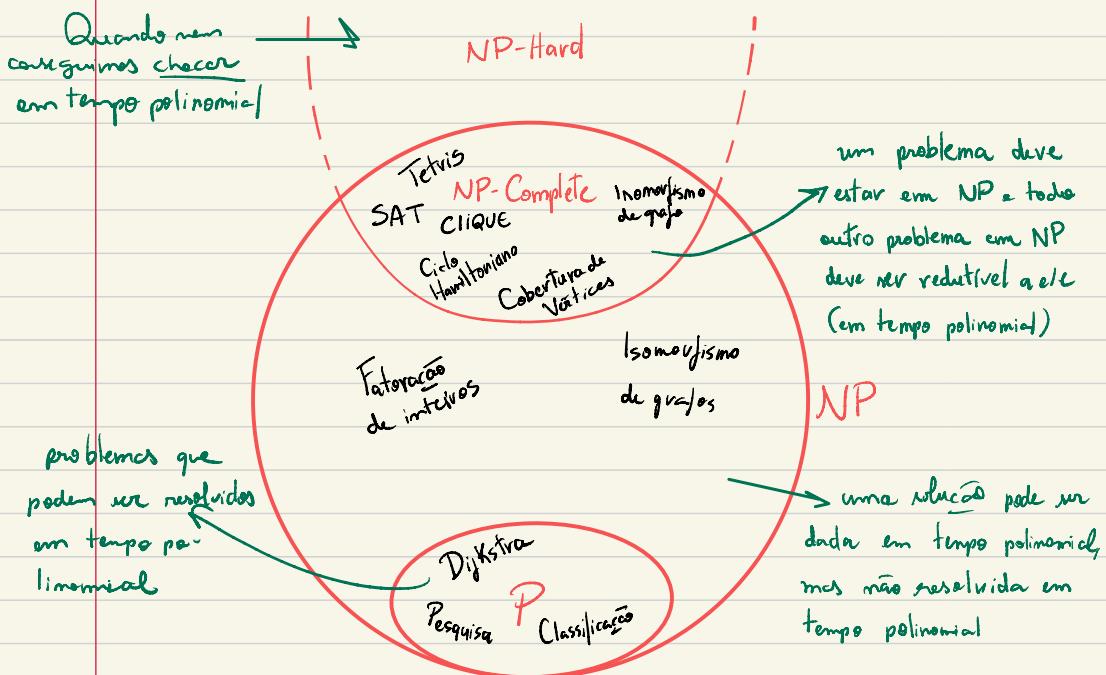
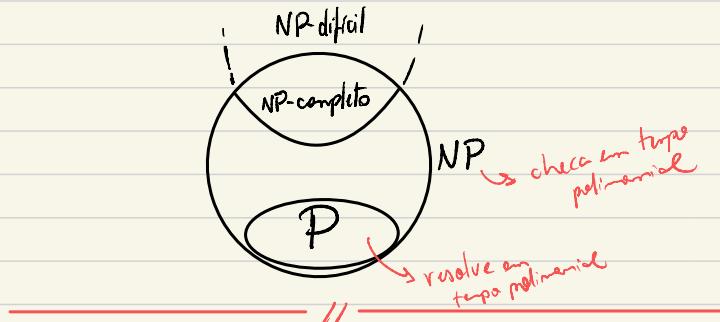
"I can't find an efficient algorithm, but neither can all these famous people."

★ um problema NP-difícil que está em NP é um problema NP-completo!

NP-Difícil:

- para um problema A se todo problema B em NP se reduz polinomialmente a A
- o NP-difícil recorre ao oráculo

Um problema A é NP-difícil se existe um algoritmo de tempo polinomial para um problema B NP-completo quando esse algoritmo tem A como oráculo;



SATISFABILIDADE (SAT):

→ dado uma fórmula booleana (V ou F) em uma forma conjuntiva normal (CNF), determina-se se existe alguma valoração p/ suas variáveis tal que ela satisfaça essa fórmula em questão.

$$\Phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_1 \vee x_2)$$

disjunção (OR)
termo/literal
conjunção (AND)
conjunção

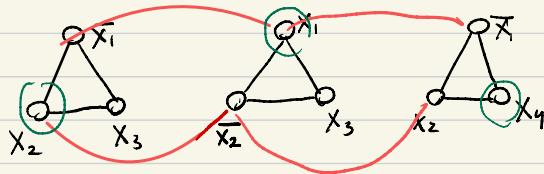
Problema 3-SAT:

→ é um problema SAT, mas cada cláusula tem 3 termos, cada um envolvendo variáveis diferentes.

Claim: $3\text{-SAT} \leq P$ Conjunto Independente

(o problema de satisfabilidade 3-SAT pode ser reduzido em tempo polinomial ao problema de Conjunto Independente)

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$



$3n^3 = O(n^3)$
n de cláusulas

$O(n^3)$
no máximo
 $O(n^2)$
 $O(n)$

- 1- create 3 vertices for each clause
- 2- connect these 3 vertices to a triangle
- 3- connect every literal to each of its negation
- 4- set K equal to number of clauses

- x_2, x_1, x_4
- x_3, x_1, x_2
- $\bar{x}_1, \bar{x}_2, x_4$

Em cada triângulo do grafo que representa uma cláusula no 3SAT, selecionamos um vértice que não seja a definição de conjunto independente (termos complementares não podem ser escolhidos, pois eles têm ligação). Se Φ é satisfatório, então existe um conjunto independente de tamanho K .



Assim, essa prova mostra que existe uma correspondência entre a satisfatibilidade da fórmula e da existência de um conjunto independente de tamanho K no grafo G .

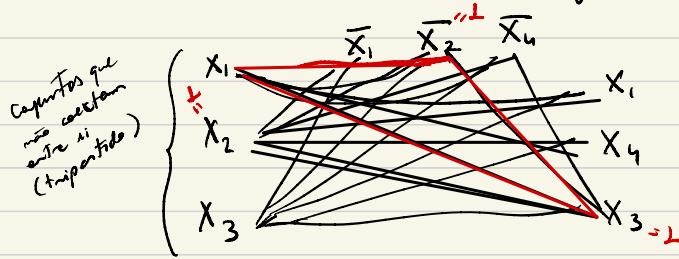
f

3-SAT \leq_p Clique \rightarrow Siga $G = (V, E)$

$(X_1 \vee X_2 \vee \bar{X}_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee \bar{X}_4) \wedge (X_1 \vee X_4 \vee \bar{X}_3) = L$

se vai existir clique
entre cláusulas diferentes
não forma conjuntos vazios em cada cláusula e conecta cada vértice aos outros se eles não negamuns uns dos outros

clique é um subgrafo conexo de G com cardinalidade $\geq K$



- unifica que cada vértice só conectado a todos os outros do subgrafo em tipo polinomial
programa NP

- **Vertex Cover:** dado um grafo $G = (V, E)$ e um número inteiro k , queremos saber se existe um subconjunto de vértices $V' \subseteq V$, com tamanho no máximo k , tal que cada aresta do grafo tenha pelo menos uma extremidade em V' .
- **Set Cover:** dados um conjunto universo U , uma coleção de subconjuntos $S_1, S_2, \dots, S_m \subseteq U$ e um inteiro k , queremos saber se existe uma subcoleção de no máximo k desses subconjuntos cuja união seja igual ao universo U .

Claim: Vertex Cover \leq_p Set Cover

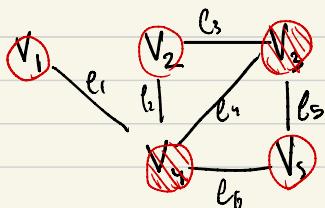
↳ (Vertex Cover pode ser reduzido polynomialmente para o Set Cover). Ou seja, vertex cover pode ser resolvido usando set cover como sub rotina, por meio de redução polynomial.

* Para reduzir Vertex Cover em Set Cover:

- 1- cadaaresta do grafo original se torna um elemento de U no Set Cover.
- 2- cada vértice do grafo original vira um subconjunto em Set Cover, que contém exatamente as arestas incidentes a esse vértice
- 3- o objetivo de cobrir todos as arestas com vértices (vertex cover) equivale a cobrir todos os elementos de U com subconjuntos (Set).

- Além, isso implica que qualquer algoritmo eficiente (polynomial) para set cover também resolveria eficientemente o Vertex Cover.
- Set Cover é um problema NP-difícil pelo menos tão complexo quanto Vertex Cover.

$$\begin{matrix} X & Y \\ \xrightarrow{\Omega(n)} & \xleftarrow{\Omega(n)} \end{matrix}$$



$$U = \{e_1, e_2, e_3, e_4, e_5, e_6\}$$

* a função \rightarrow é inversa
(intensidade $\Omega(g(n))$), $\text{mono } f(n)$

$$S_1 = \{e_1\}$$

$$S_2 = \{e_2, e_3\}$$

$$S_3 = \{e_3, e_4, e_5\} \leftarrow$$

$$S_4 = \{e_4, e_2, e_3, e_6\} \leftarrow$$

$$S_5 = \{e_5, e_6\}$$

tempo
polynomial

$$S_3 \cup S_4 = U \mid R=2$$

$\Omega(n^2)$

NP \rightarrow usar uma hash prévia
com resposta (nível de NP) $n \cdot m$

* todos instâncias de Vertex Cover vai dar certo em Set Cover, mas não é o contrário.

* como Y é tão difícil quanto X, então é NP-completo.

Relações de Recorrência

expandir: usa repetidamente a recorrência para expandir a expressão a partir do n -ésimo termo até o caso base.

conjecturar: a expansão nos guia a conjecturar a solução de recorrência

verificar: a conjectura é finalmente verificada por indução

* Exemplo:

- expandindo → aplicando a definição de n para $n-1, n-2\dots$

$$\begin{aligned} \triangleright S(n) &= 2S(n-1) \\ &= 2[2S(n-2)] = 2^2S(n-2) \\ &= 2^2[2S(n-3)] = 2^3S(n-3) \\ &= 2^3[2S(n-4)] = 2^4S(n-4) \end{aligned}$$

↓ após K expansões, a equação tem a forma:

$$S(n) = 2^K S(n-K)$$

- a expansão deve parar no caso base ($n-K=L$) $\rightarrow K=n-L$

$$\begin{aligned} S(n) &= 2^{n-L} S[n-(n-L)] \\ &= 2^{n-L} S(L) \\ &= 2^{n-L}(2) \\ &= 2^n \end{aligned}$$

- a base da indução é $S(1) = 2^1$, que é verdade pelo caso base da definição recursiva.
- supõe-se que $S(K) = 2^K$, queremos provar que $S(K+L) = 2^{K+L}$

$$S(K+L) = 2S(K)$$

ainda, provar-se $S(K+L) = 2(2^K)$
a relação de recorrência. $\rightarrow S(K+L) = 2^{K+L}$

Fórmulas (talvez) necessárias para os questões de R.R.:

★ Progressão Geométrica →

$$S_n = \frac{a + (r^n - 1)}{r - 1}$$

★ Propriedades dos Logaritmos:

↳ definição → $x = b^a \iff \log_b x = a$

↳ mudança de base → $\log_b a = \frac{\log_c a}{\log_c b}$

↳ potência de um logaritmo → $(\log_b a)^n = \log_b^n a$

↳ logaritmo de base c/ potência → $\log_a x b = \frac{b}{x} \log_a b$

Logaritmos

Definición:	Exponente:
$\log_b x = a \Rightarrow x = b^a$	$x^{\log_b a} = a^{\log_b x}$
Misma base y argumento:	Potencia del Argumento:
$\log_b b = 1 ; b > 0, b \neq 1$	$\log_b a^n = n \cdot \log_b a$
Argumento Uno:	Raíz del Argumento:
$\log_b 1 = 0 ; b > 0, b \neq 1$	$\log_b \sqrt[n]{a} = \frac{1}{n} \cdot \log_b a$
Producto:	Potencia idéntica de Base y Arg.
$\log_b(x \cdot y) = \log_b x + \log_b y$	$\log_b^n a^n = \log_b a$
Cociente:	Cambio de base por argumento:
$\log_b \left(\frac{x}{y} \right) = \log_b x - \log_b y$	$\log_b a = \frac{1}{\log_a b}$
Cambio de base:	Potencia de un logaritmo:
$\log_b a = \frac{\log_c a}{\log_c b}$	$(\log_b a)^n = \log_b^n a$

Questões de R.R. que acho mais importantes:

$$\star \begin{cases} T(n/2) + C, n > L \\ C, n = L \end{cases}$$

↓ expansão telescópica

$$T(n) = T(n/2) + C$$

$$T(n/2) = T(n/4) + C$$

⋮

$$T(n/2^i) = T(n/2^{i+1}) + C$$

✓

$$\frac{n}{2^i} = n \Rightarrow i=0$$

$$\frac{n}{2^1} = 1 \rightarrow n = 2^1$$

log₂

n = 1

quando o valor final de i é considerado como 2, temos que somar o caso base na somatória!!!

mais caso para ex, não tem necessidade de

porém no T(2), já que

mais caso das repetições

não temos os termos

(C). Entretanto, temos que adicionar ao somatório acima o caso base T(2).

$$\sum_{i=0}^{\log_2 n} C$$

log₂ n - 0 + 1 vezes

$$\text{fórmula fechada: } T(n) = C(\log_2 n - 0 + 1)$$

$$T(n) = C(\log_2 n + 1)$$

$$C \cdot \log_2 n + 1 \leq K \cdot \log n$$

$$\star T(n) = \begin{cases} 2T(n/2) + C, n > 1 \\ C, n = 1 \end{cases}$$

$$T(n) = 2T(n/2) + C$$

$$2T(n/2) = 2 \cdot 2T(n/4) + C$$

$$2^i T(n/2^i) = 2^i \cdot 2T(n/2^{i+1}) + C$$

$$P_G = \frac{a_1(r^n - 1)}{r - 1}$$

$$T(2) = 2T(1) + C$$

$$T(1) = 2^1 C$$

$$\frac{1}{2^1} = n = 1 \Rightarrow 2^1 C$$

$$\frac{n}{2^1} = 1 \Rightarrow \log_2(n)$$

$$\sum_{i=0}^{\log_2 n} 2^i \cdot C$$

$$T(n) = C \cdot \left(\frac{2^{\log_2 n + 1} - 1}{2 - 1} \right)$$

$$\frac{2n - 1}{1}$$

$$O(n) \Leftrightarrow 2Cn - C$$

sobreindo n.

$$T(n) = C(2n - 1)$$

$$C(2n - 1) \leq C \cdot n$$

analisando assimétricamente, a multiplicação da constante mais influente no resultado