Anotações MC658

Eduardo M. F. de Souza

11 de abril de 2020

Sumário

1 Classes de Complexidade

1.1 Características

- São classes que contém problemas, e, problemas que contém determinadas características em comum;
- A maior parte dos algoritmos vistos até então têm tempo polinomial — $O(n^k)$ onde k é constante e n é o tamanho da entrada;
- Nem todo problema admite um algoritmo polinomial para resolvê-lo.

  Exemplo: problema da parada, no qual sequer admite um algoritmo, independente do tempo; programa para predizer se um algoritmo pode entrar em *deadlock* ou não em uma máquina genérica;

  - É **tratável** se admite um algoritmo polinomial;
  - E **intratável** se não admitir um algoritmo polinomial (pode admitir um algoritmo exponencial);

1.2 $\mathcal{P}$, $\mathcal{NP}$ e $\mathcal{NP}$-*Completo*

- $\mathcal{P}$**:** Classe em que os problemas que possuem algoritmos que os resolvem em tempo polinomial;
- $\mathcal{NP}$**:** Classe em que os problemas admitem um algoritmo polinomial que verifiquem instâncias do problema.

  Um exemplo é o Ciclo Hamiltoniano:

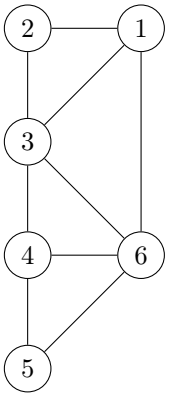
  Entrada: Grafo simples $G = (V, E)$.

  Saída: Existe ou não um ciclo que passa por cada vértice exatamente uma vez.

  Verificação:

  (1,2,3,6,4,5) → não é uma solução válida;

  (1,2,3,4,5,6) → é uma solução válida;



- $\mathcal{NP}$***-Completo*****:** Todo problema desta classe está em NP.Problemas NPC têm a característica de, se algum deles admitir um algoritmo polinomial, então automaticamente todos os problemas de NP possuem um algoritmo polinomial;

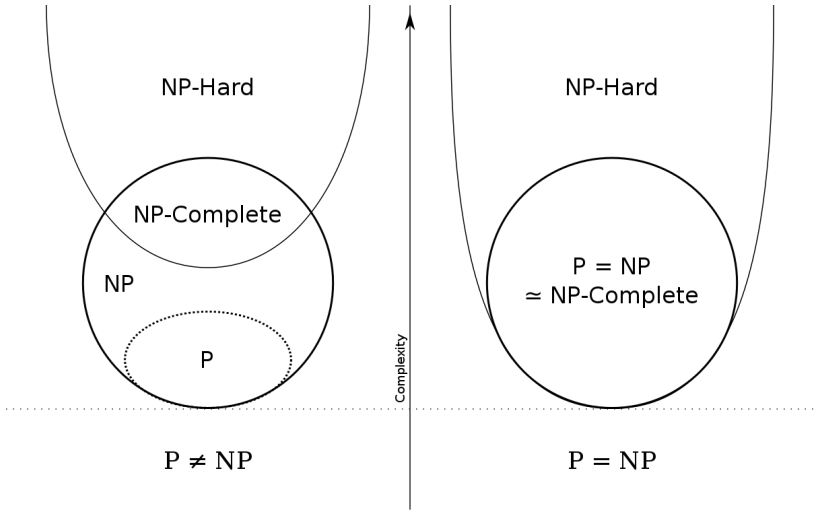
1.3 Problemas de Decisão

Resposta deve ser de **Sim (1)** ou **Não (0)**.

Exemplo: Da versão de decisão do problema do menor caminho em um grafo:

Entrada: $G = (V, E)$ com pesos nas arestas, valor k, e $u, v \in V$

Pergunta: Existe caminho $u \to v$ com custo $\leq k$?



$\mathcal{P} \leq \mathcal{NP} \mid \mathcal{NP} \in \mathcal{P}$ é uma questão em aberto. Desde 1970 até hoje um importante problema aberto é se $\mathcal{P} = \mathcal{NP}$.

1.3.1 Formalização matemática de um problema

- Um problema será definido como um lingagem sobre um alfabeto Σ;
- Usaremos $\Sigma = 0, 1$ (binário);
- Dado um problema Q, qualquer instância desse problema será codificada como uma string de 0s e 1s;
- Uma linguagem $\underline{L}$ de Σ é um conjunto de strings formadas com os símbolos de Σ.
  Notação:
  - E: string vazia
  - $\emptyset$: Uma linguagem vazia
  - Σ^*: Todas as possíveis strings que podem ser escritas com os símbolos de Σ.
    Exemplo: $\Sigma^* = \{E, 0, 1, 00, 01, \ \ldots\}$

Observação: Qualquer lingaguem L é tal que L contem em Σ^*.

Um problema de decisão Q será visto como uma linguagem que contém todas as strings de Σ^* que corresponde às instâncias de Q cuja resposta é sim.

Exemplo:

Ciclo-Hamiltoniano = $\{ \langle G = (V, E) \rangle$ tal que G possui um Ciclo Hamiltoniano$\}$

$\langle \ \rangle$: Notação que corresponde à codificação da instância como uma string.

1.3.2 Operações sobre Linguagens

- **União:** $L_1 \cup L_2 = \{s : s \in L_1 \text{ ou } s \in L_2\}$
- **Concatenação:** $L_1 L_2 = \{x, y : x \in L_1 \text{ ou } y \in L_2\}$
- **União:** $L_1 \cap L_2 = \{s : s \in L_1 \text{ ou } s \in L_2\}$

1.4 Algoritmos

- Uma entrada para um algoritmo A é qualquer string de Σ^*;
- Um algoritmo aceita $s \in \Sigma^*$ se $A(s) = 1$;
- Um algoritmo rejeita $s \in \Sigma^*$ se $A(s) = 0$;
- Um algoritmo aceita uma linguagem L se $\forall s \in L \Rightarrow A(s) = 1$. Para strings $s \notin L \Rightarrow A(s) \neq 1$ o algoritmo pode não parar — entra em loop infinito;
- A é um algoritmo que decide uma linguagem L se $\begin{cases} \forall s \in L \Rightarrow A(s) = 1 \\ \forall s \notin L \Rightarrow A(s) = 0 \end{cases}$
- Uma linguagem é **aceita** em tempo polinomial se existe um algoritmo polinomial A que **aceita** L;
- Uma linguagem é **decidida** em tempo polinomial se existe um algoritmo polinomial que **decide** L;

1.5 Definição da classe $\mathcal{P}$

$$\mathcal{P} = \{L \in \Sigma^* \mid L \text{ é decidido em tempo polinomial}\}$$

Teorema: $\mathcal{P}$ é o conjunto de todas as linguagens decididas em tempo polinomial.

$$\mathcal{P} = L \in \Sigma^* \mid L \text{ é aceita em tempo polinomial} = \mathcal{P}'$$

Prova:

Contido:

$$\text{Seja } L \text{ decidida em tempo polinomial} \Rightarrow$$
$$\Rightarrow \exists \text{ um algoritmo } A \text{ polinomial que decide } L \Rightarrow$$
$$\Rightarrow A \text{ aceita } L \text{ em tempo polinomial;}$$

Contrário de contido:

$$\text{Seja } L \text{ aceita em tempo polinomial} \Rightarrow$$
$$\Rightarrow \exists \text{ um algoritmo } A : s \in L \mid A(s) = 1 \text{ em tempo } c \times |s|^k \text{ (polinomial) onde } c \text{ e } k \text{ são constantes;}$$

Existe um algoritmo A' que simula A por $c \times |s|^k$ passos. Se $A(s) = 1$, então $A'(s) = 1$ e caso $A(s)$ não responda nada (ou zero) então $A'(s) = 0 \Rightarrow L$ é decidida em tempo polinomial.

Problema de Decisão

$$\Sigma = \{0, 1\}$$
$$\Sigma^* = \{E, 0, 1, 00, 01, \ldots\}$$

Problema $Q \subseteq \Sigma^* \mid x \in Q \iff$ a resposta para instância x é sim.
Algoritmo A que decide uma linguagem $L \subseteq \Sigma^*$:

$$A(x) = \begin{cases} 1, \ \forall x \in L \\ 0, \ \forall x \notin L \end{cases}$$

$$P = \{L \subseteq \Sigma^* \mid \exists \text{ um algoritmo } A \text{ que decide } L \text{ em tempo polinomial}\}$$

1.6 Verificação

Ciclo Hamiltoniano = {$<G = (V, E) >$ onde G é um grafo simples e G possu um ciclo que passa por cada vértice exatamente uma vez

Algoritmo Verificador para uma linguagem $L \subseteq \Sigma^*$. $A(x, y)$ onde x é uma instância de L e y é outra string que chamamos de certificado.

1. Se $x \in L$ então $\exists y \in \Sigma^* / A(x, y) = 1$
2. Se $x \notin L$ então $\forall y \in \Sigma^* / A(x, y) \neq 1$

1.7 Algoritmo Verificador para Ciclo Hamiltoniano

A(G, sequência vertical $(v_1, v_2, \ldots, v_n)$) -> Verificar que correspondem a todos os vértices do grafo exatamente uma vez. For v $\in (v_1, v_2, \ldots, v_n) If v \notin G return 0 For i = 1 to n - 1 if (v_i, v_{i+1}) \notin G return 0 if (v_1, v_n) \not in G return 0 return 1$

Se G $\in Ciclo-Hamiltoniano, entoexisteumasequnciadevrticasquecorrespondeaocicloeusamosistocomocertificad$

Se G $\not in Ciclo-Hamiltoniano entonachecagemdasarestas, paraqualquersequnciadevrtices, umaarestaestar faltan$

Um algoritmo A(x,y) verifica L $\subseteq \Sigma^* em tempo polinomial se A executa em tempo polinomial em |x|e|y|e$:

Se x $\in L, ento \exists y \in \Sigma^* / |y| = O(|x|^k) para k constante. A(x, y) = 1 Se x \notin L ento \forall y \in \Sigma^* / |y| \ in O(|x|^k) temos que A(x, y$
1

1.8 Classe NP

NP = $\{L \subseteq \Sigma^* / \exists A(x, y) polinomial que verifica L\}$

Questão em aberto: P = NP?

Exercício> P $\subseteq NP$

Seja L $\in P (mostrar que L \in NP) Existe algoritmo A(x) que decide L em tempo polinomial.$

A1(x,y) return A(x)

1. $x \in L$, então usando $y = E$ temos que $A'(x, y) = 1$ e $|y| \in O(|x|^k)$
2. $x \notin L$, então $A(x) = 0 \rightarrow A1(x, y) = 0$ independente do y.

1.9 Classe co-NP

Dado $L \subseteq \Sigma^*$ definimos o complemento de l como

$$\overline{L} = \Sigma^* \backslash L$$

co-NP = $\{L \subset \Sigma^* / \overline{L} \in \text{NP}\}$

Instintivamente, co-NP contém as lingaguens para as quais existe algoritmo verificador polinomial para instâncias "não"do problema. $L \in$ co-NP se existe um algoritmo polinomial $A(x, y)$, dois quais x é instância e y é um certificado, onde:

1. $x \not\in nL, \exists y \in \Sigma^* / |y| = O(|x|^k)$ e $A(x, y) = 1$
2. $x \in L, \forall y \in \Sigma^* / |y| = O(|x|^k)$ e $A(x, y) \neq 1$

Primos { <n> N é um inteiro positivo e é primo}

Primos $\in co{-}NPpoispodemoscriarumalgoritmoqueparacadan \neq Primos \rightarrow \exists umdivisorddenqueservecomocertificaa$

Questão em aberto> NP = co-NP ?

Exercício: p $\subseteq NP \cap co - NP(AcabamosdeverqieP \subseteq NP)MostrarP \subseteq co - NP!SejaL \in P \rightarrow existeumalgoritmopolinomialA(x)quedecideL.$

A'(x,y) if A(x) == 1 return 0 if A(x) == 0 return 1

1. $x \notin L$ então A(x,E) = 1 em tempo polinomial
2. $x \in L$ então A(x,y)=0 independe do y.

1.10 Possíveis Relações entre estas Classes

P = NP $\rightarrow P = NP = co - NPP \neq NP \rightarrow P \in NP = co - NPP \neq NP \rightarrow P = NP \bigcap co - NP, NP \neq co - NPp \neq NP \rightarrow P \subset NP \bigcap co - NP, NP \neq co - NP$

$n = p_1^{j_1} \times p_2^{j_2} \times \ldots \times p_q^{j_q}$ Qualquer $n \in^+$ possui uma fatoração única em primos distintos.

RSA (ninguém consegue achar um fator de n em tempo polinomial)

Fatoração: Dado n e m $\leq nexisteumfatorpdentalque$p $\leq m$

Fatoração $\in P?Ningumsabe.$

Fatoração $\in co - NPeFatorao \in NP.$

Fatoração $\in NP : Dado(n, m) \in Fatorao, bastausarcomocertificadoumfatop \leq m!Fatorao \in co - NP : (n, m) \notin Fatorao \rightarrow fatorprimop \leq mtalquepdividan.Meucertificadoumafatoraodenemprimos$p$_1^{j_1}, p_2^{j_2}, \ldots, p_q^{j_q}$

1. Verificar que a multiplicação do valor n n = ppp
2. Verificar que cada p$_j umnmeroprimo(feitoemtempopolinomialcomAKSde2006)Verificarquecadap_j \geq m$

1.11 Reduções Polinomiais

Definição: sejam $L_1, L_2 \subseteq \Sigma^*$ duas linguagens. Dizemos qye L_1 se reduz para L_2 em tempo polinomial se:

3. Existe um algoritmo F que tranforma a instância x_1 de L_1 em instâncias $x_2 = F(x_1)$ de L_2 em tempo $O(|x_1|^k)$, k constante;
2. $x_1 \in L_1 \iff F(x_1) = x_2 \in L_2$;

Pergunta: Dado $x_1 \notin L_1$, é possível $F(x_1) = x_2 \in L_2$? **R:** Não é possível!

1.11.1 Lema 34.3

Sejam $L_1, L_2 \subseteq^*$ tal que $L_1 \leq_p L_2$. Se $L_2 \in P$, então $L_1 \in P$

Como $L_2 \in P \rightarrow \exists$ algoritmo polinomial A_2 que decide L_2. Podemos construir: $\mathrm{A}_1(x_1) : x_2 = F(x_1) return A_2(x+2)$

Como $L_1 \leq_p L_2$ existe o algoritmo de redução F de L_1 pra L_2.

- A_1 executa em tempo polinomial (tanto A_2 quanto F têm tempo polinomial)
- 1. Dado $x_1 \in L_1 \rightarrow F(X_1) = x_2 \in L_2$ e $A_2(x_2) = 1 \rightarrow A_1(x_1) = 1$
  2. Dado $x_1 \notin L_1 \rightarrow F(x_1) = x_2 \notin L_2$ e $A_2(x_2) = 0 \rightarrow A_1(x_1) = 0$

1.12 NP-Completo (NPC)

Uma linguagem $L \in$ NPC se ela satisfizer:

1. $L \in$ NP
2. $\forall L' \in$ NP então existe redução polinomial de L' para L, $L1 \leq_p L$

Obs: As linguagens (problemas) que só satisfazem a condição (2) pertencem à classe $\mathrm{NP}_D ifcil$.

1.13 Teorema 34.4

Seja $L \in$ NPC. Se $L \in$ P, então $\forall L' \in$ NPC. Temos que $L' \in$ P

Prova: Seja $L' \in$ NP um problema qualquer. Sabemos que $L \in$ NPC $\rightarrow L' \leq_p L$.
Sabemos que $L \in$ P, então, pelo lema anterior, $L' \in$ P!

$$P_1 \in \text{NPC}$$
$$\forall L \in \text{NP},\ L \leq_p P_1$$
$$P_2,\ P_1 \leq_p P_2$$

Ao mostrarmos que um problema $L_1 \in NPC$, estamos dando fortes indícios que L_1 não admite um algoritmo polinomial. Melhor resolver L_1 com técnicas para lidar com problemas NP-Difíceis.

1.14 Teorema de Cook (1971)

1.14.1 Problema SAT

Dadas variáveis booleanas $x_1, \ldots, x_n$ e uma fórmula sobre elas com operadores $\vee$, $\wedge$, $\neq$, $\rightarrow$, $\iff$, existe uma atribuição para $x_1, \ldots, x_n$ tal que f fica verdadeira?

1.14.2 Circuit-SAT

Portas lógicas: not, or, and (podem ter mais entradas).

Dado um circuito lógico com entradas $x_1, \ldots, x_n$ e uma única saída, existe uma atribuição para $x_1, \ldots, x_n$ tal que a saída do circuito é verdadeira?

Os circuitos considerados não possuem *loop*?

1.14.3 Circuit-SAT $\in$ NP

Vamos mostrar que Circuit-SAT é NP-Difícil.

Uma máquina é construída com vários circuitos lógicos.

Seja $L \in$ NP, queremos mostrar que existe $L \leq_p$ Circuit-SAT. Sabemos que existe um algoritmo $A(x, y)$ verificador polinomial para L.

1. Se $x \in L$, $\exists y$ tamanho polinomial tal que $A(x, y) = 1$;
2. Se $s \notin L$, $\forall y$ tamanho polinomial tal que $A(x, y) \neq 1$;

Dado x, assumimo que A executa no máximo $c_1|x|^{k_1}$ passos (c_1, k_1 são constantes) e $|y| \leq c_2|x|^{k_2}$, onde c_2 e k_2 são constantes.

Podemos üsarö computador para executar o algoritmo A

Dado x uma instância de L. Montamos um circuito l[ogico com $c_1|x|^{k_1} A$ cópias do c̈omputadorq̈ue faz a simulação do algoritmo verificador A; Tamanho dos circuitos representando, A, PC, Mem e Controle são constantes. Assumimos que x é setado fixo com seu próprio valor. A única entrada deste circuito é o y; Dado $x \in \Sigma^*$ instância de L, construímos um circuito C em tempo polinomial!

1. Se $x \in L$ então $\exists y$ polinomial tal que $A(x, y) = 1$. Este mesmo y serve como entrada para o circuito C, deixando ele satisfazível $\rightarrow C \in$ Circuit-SAT
2. Se $x \not\in L$ então $\forall y$ de tamanho polinomial, $A(x, y) \neq 1 \forall$ entrada y de C $\rightarrow Cnuncasersatisfazvel \rightarrow C \notin$ Circuit-SAT.

1.15 Provas de NP-Completude

Mostramos que o Circuit-SAT é NP-Completo:

1. Circuit-SAT $\in$ NP;
2. $\forall L \in$ NP existe redução em tempo polinomail de L / leq_{pol} Circuit-SAT.
   $x \in L \iff C \in$ Circuit-SAT

Vale a transitividade para $\leq_{pol}$: $P_1 \leq_p olP_2$ e $P_2 \leq_p olP_3 \rightarrow P_1 \leq_p olP_3$

x_1	x_2	$x_1 \to x-2$	$x_1 \iff x_2$
0	0	1	1
0	1	1	0
1	0	0	0
1	1	1	1

1.15.1 SAT

Fórmula booleana f com variáveis $x_1, \ldots, x_n$ e operadores $\neg$, $\vee$, $\wedge$, $\to$, $\iff$. Existe atribuição para $x_1, \ldots, x_n$ tal que f fica verdadeiro.

$$f = (x_1 \to x_2) \vee \neg((\neg x_1 \iff x_3) \vee x_4) \wedge \neg x - 2$$

SAT = $\{ < f >$ onde f é uma fórmula lógica que possui atribuição verdadeira$\}$

Para mostrar que SAT é NP-Difícil (condição 2 de NPc) faremos Circuit-SAT $\leq_{pol}$ SAT $\forall L \in$ NP, sabemos que $L \leq_{pol}$ Circuit-SAT e, por transitividade, teremos $L \leq_{pol}$ SAT.

Teorema 34.9: SAT $\in$ NP-Completo. Prova:

1. SAT $\in$ NP (fica como exercício);
2. Mostrar que Circuit-SAT $\leq_{pol}$ SAT:

   $C \in$ Circuit-SAT $\iff$ $f \in$ SAT.

   Dado um circuito C qualquer, vamos construir a fórmula f em tempo polinomial onde vale o item anterior.

   Dado c além das variáveis de entrada, criamos uma nova variável para saíde de uma porta lógica do circuito.

   Escreveremos uma cláusula para cada porta lógica.
   Exemplo: $(x_5 \iff (x_1 \vee x_2))$

   Cada cláusula só pode ser verdadeira quando o valor de variável de saída da porta lógica correspondem ao que é computado pela porta lógica.

   A formula f será um *and* de todas as cláusulas correspondentes a cada uma das portas lógicas mais a última variável de saída do circuito.

$$\begin{aligned} f = x_{10} \wedge (x_4 &\iff (\neg x_3)) \\ \wedge (x_5 &\iff (x_1 \vee x_2)) \\ \wedge (x_6 &\iff \neg x_4) \\ \wedge (x_7 &\iff (x_1 \wedge x_2 \wedge x_4)) \\ \wedge (x_8 &\iff (x_5 \vee x_6)) \\ \wedge (x_9 &\iff (x_6 \vee x_7)) \\ \wedge (x_{10} &\iff (x_7 \wedge x_8 \wedge x_9)) \end{aligned}$$

Dado C qualquer podemos construir f em tempo proporcional ao número de portas lógicas em C e, portanto, em tempo polinomial.

1. Suponha que $C \in$ Circuit-SAT:

   $\rightarrow Existeumaatribuioparaasvariveisdeentradaquedeixa$Cverdadeiro.$Simulandoessaentradanocircuito,$ $cadaclusulacorrespondeaumaportalgicaficaverdadeira.$

Além disso, a saída de C é verdadeira $\rightarrow avariveldesadafinalverdadeira \rightarrow$f$ficaverdadeiro.f \in$ SAT

$f \in$ SAT (temos que mostrar que C que deu origem a f é tal que $C \in$ Circuit-SAT)
$f \in$ SAT $\rightarrow existeatribuioparavarivelkde$f$quedeixafverdadeiro.Usamososvaloresdasvariveisdeentradaemfcomoaentra$
$Paracadaportaportalgicade C, oseuvalordesadaigualaovalordavariveldesadadaclusulacorrespondenteaessaportalgica$
$Ovalordesadadecadaclausulaigualaportalgicacorrespondentequandodamosessaentrada.$
$Comoemfasadafinalverdadeiraeeladeveserigualasadadaltimaportalgica \rightarrow Asadadocircuito1 \rightarrow$C $\in$ Circuit-SAT.

1.16 3 CNF-SAT

Uma fórmula que é uma conjunção (and) de cláusulas e cada cláusula é uma disjunção (or) de exatamente três literais (x_i ou $\overline{x_i}$).
Existe atribuição verdadeira?A
Exemplo: $f = (x_1 \vee \overline{x_2} x_3) \wedge (x_4 \vee \overline{x_5} \vee x_1) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$

$\boxed{Teorema : 3CNF - SATNP - Completo}$ Prova:

1. 3CNF-SAT $\in$ NP (exercício)
2. 3CNF-SAT $\in$ NP-Difícil.
   Faremos SAT $\leq_{pol}$ 3CNF-SAT. A partir de f_1 de SAT, construiremos um f_2 do 3CNF-SAT.

$$f_1 - ((x_1 \rightarrow) \wedge \neg((\neg x_1 \iff x_3) \wedge \neg x_2$$

   Construímos uma árvore de avaliação da fórmula onde as folhas são os literais da fórmula, nós são os operadores lógicos da fórmula.

   Criamos f_1' equivalente a f_1 que corresponde à avaliação da árvore construída.

$$\begin{aligned} f_1' = y_1 \wedge (y_1 &\iff (y_2 \wedge \neg x_2)) \\ \wedge (y_2 &\iff (y_3 y_4)) \\ \wedge (y_3 &\iff (x_1 \rightarrow x_2)) \\ &\vdots \\ \wedge (y_5 &\iff \neg x_6) \end{aligned}$$

   Cada cláusula em f_1 tem no máximo 3 literais (cada operador lógico tem no máximo 2 entradas e tem uma saída);

   f_1' pode ser computado em tempo polinomial.

   Deixar cada cláusula na CNF

Para cad cláusula f_i' de f_1, construímos uma tabela verdade.

Exemplo: $(y_1 \iff (y_2 \wedge \overline{x_2})) = f_i'$

POdemos escrever fórmulas na DNF (forma normal disjuntiva equivalente à f_1'

$$f_i'' = (\overline{y_1} \wedge \overline{y_2} \wedge \overline{x_2}) \vee (\overline{y_1} \wedge \overline{y_2} \wedge x_2) \vee \ldots$$

$\boxed{Criar \overline{f_i''} \text{ na DNF que corresponde a } f_i}$

$$\overline{f_i'} \approx \overline{f_i''} = (\overline{y_1} \wedge y_2 \wedge \overline{x_2}) \vee (y_1 \wedge \overline{y_2} \wedge \overline{x_2}) \vee (y_1 \wedge \overline{y_2} \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2))$$

Aplicando De'Morgen, obtemos:

A partir de f_1', construímos f_1'' equivalente e que está na CNF e isso em tempo polinomial.

Cada cláusula de f_1' temos no máximo 3 variáveis $\rightarrow tabelaverdadecomnomximo8entradas.$

$\rightarrow nmerodeclusulasnomximo8vezesonmerooriginaldeclusulasemf_1'$

Deixar cada cláusula de f_1'' com 3 literais. Suponha uma cláusula de 2 literais $(l_1 \vee l_2) \iff (l_1 \vee l_2 \vee z) \wedge (l_1 \vee l_2 \wedge \overline{z})$