UDESC

Complexidade de Algoritmos

Prof. Rafael Alceste Berri
rafaelberri@gmail.com

Prof. Diego Buchinger
diego.buchinger@outlook.com

Prof. Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

UDESC

Reduções de Problemas

UDESC

NP-Completo

Um problema X é ***NP-Completo*** se:

1. O problema deve ser NP: $X \in NP$

   a) Conseguir um algoritmo não determinista que resolva o problema em tempo polinomial

   b) Conseguir um algoritmo determinista que verifica em tempo polinomial se uma resposta é verdadeira ou não ***(certificado)***

2. Fazer a redução de um problema NP-Completo (Y) conhecido para o problema X: $Y \leq_p X$ para todo $Y \in NP$

UDESC

Problemas intratáveis

Tempo Assint.	n	n log n	n^2	n^3n3	2^n
n=10	0,00033s	0,0015s	0,0013s	0,0034s	0,001s
n=100	0,003s	0,03s	0,13s	3,4s	4 x 10^{14}
n=1.000	0,033s	0,45s	13s	0,94h	Séculos
n=10.000	0,33s	6,1s	22m	39 dias	...
n=100.000	3,3s	1,3m	1,5 dias	108 anos	..
1s -> n máximo	3 x 10^4	2000	280	67	20
1m -> n máx	18 x 10^5	82000	2200	260	26

(SAT) Satisfatibilidade de Fórmulas Booleanas

O problema da ***Satisfatibilidade de fórmulas booleanas*** consiste em determinar se existe uma atribuição de valores booleanos, para as variáveis que ocorrem na fórmula, de tal forma que o resultado seja ***verdadeiro***.

Um ***literal*** é uma variável proposicional ou sua negação.

Exemplo:

$$x_1 \wedge (x_2 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

Problema de Decisão: existe uma combinação de valores para x_1 e x_2 que satisfazem esta equação?

Complexidade algoritmo trivial: (2^n)

(SAT) Satisfatibilidade de Fórmulas Booleanas

Classificando SAT como NP-Completo:

Passo 1: Algoritmo de certificado (determinista é polinomial)

Passo 2: MTND $\leq_p$ SAT

```
bool certificado( bool *sol ){
   return sol[1] &&
          (sol[2] || !sol[1]) &&
          (!sol[2] || !sol[3]) &&
          (!sol[1] || sol[2] || sol[3]);
}
```

$$x_1 \wedge (x_2 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

(SAT) Satisfatibilidade de Fórmulas Booleanas

// **Algoritmo SAT Determinista** -> O(2^n)

```
Procedure Aval(E,n)
Begin
    For x1 <- 0 to 1 do
        For x2 <- 0 to 1 do
            ....
            For xn <- 0 to 1 do
                if E(x1,x2,...xn) = true then
                    sucesso
end
```

UDESC

(SAT) Satisfatibilidade de Fórmulas Booleanas

```
// Algoritmo SAT Não Determinista -> O(n)
Procedure Aval(E,n)
Begin
    For i <- 1 to n do
        xi <- Escolhe(true,false)

    if E(x1,x2,...xn) = true then
        sucesso
    else
        insucesso
end
```

UDESC

NP-Completo

Teorema de Cook(-Levin): SAT é um problema NP-Completo

SAT está em P se, e somente se, P = NP

qualquer problema em NP pode ser reduzido em tempo polinomial por uma máquina de Turing não determinista a um problema SAT.

$$\text{MTND} \leq_p \text{SAT}$$

Não vamos fazer essa redução pois ela é mais longa

http://www.inf.ufrgs.br/~prestes/Courses/Complexity/aula27.pdf

https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem

UDESC

(SAT) Satisfazibilidade de Fórmulas Booleanas

Classificando SAT como NP-Completo:

Passo 1: Algoritmo de certificado (*passed*)

Passo 2: MTND $\leq_p$ SAT (*passed*)

Logo, provamos que SAT pertence ao conjunto de problemas NP-Completo!

Forma Normal Conjuntiva

Uma formula booleana está na ***Forma Normal Conjuntiva*** (***CNF***) se é expressa por um grupo cláusulas AND, cada uma das quais formada por OR entre literais.

Uma fórmula booleana esta na ***k-CNF*** se cada cláusula possui exatamente *k* literais:

Exemplo 2-CNF:

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$$

Não é NP-Completo!

3-CNF-SAT

Problema: verificar se uma fórmula booleana na 3-CNF é satisfazível.

3-CNF-SAT é *NP-Completo*?

- **Passo 1**: 3-CNF-SAT $\in$ *NP*.
- **Passo 2**: SAT $\leq_p$ 3-CNF-SAT.

Exemplo de instância 3-CNF-SAT:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$$

3-CNF-SAT

Passo 1: 3-CNF-SAT $\in$ *NP*.

```
bool certificado( bool *sol ){
   return ( sol[1] ||  sol[2] ||  sol[3] )
        && (!sol[1] || !sol[2] || !sol[3] )
        && ( sol[1] ||  sol[2] || !sol[3] );
}
```

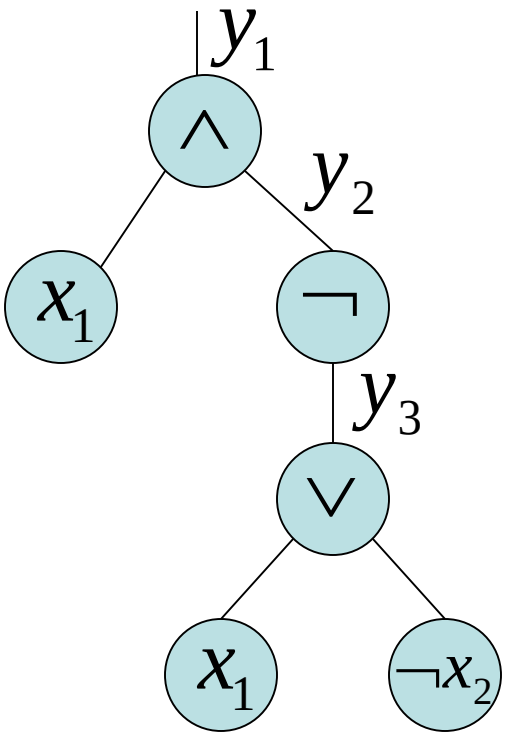
$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$$

$SAT \leq_p$ 3-CNF-SAT

Instância

Dada uma fórmula booleana: $\phi = x_1 \wedge \neg(x_1 \vee \neg x_2)$

SAT

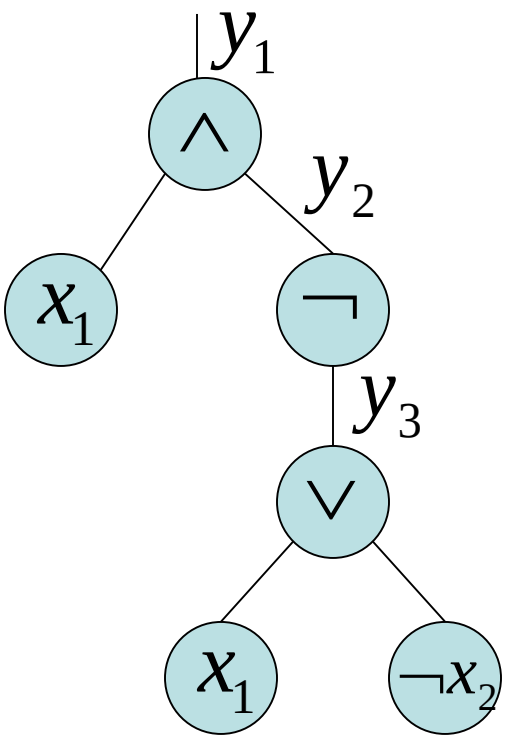


REDUÇÃO

1. Construir uma árvore que represente à fórmula.
2. Introduzir uma variável y_i para a raiz e a saída de cada no interno.

$SAT \leq_p$ 3-CNF-SAT

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow \neg y_3) \wedge (y_3 \leftrightarrow (x_1 \vee \neg x_2))$$



3. Reescrevemos a fórmula original como conjunções entre a variável raiz e as cláusulas que descrevem as operações de cada nó.

Introduz **uma** variável e **uma** cláusula para cada operador.

$SAT \leq_p$ 3-CNF-SAT

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow \neg y_3) \wedge (y_3 \leftrightarrow (x_1 \vee \neg x_2))$$

4. Para cada ϕ'_i construir uma tabela verdade, usando as entradas que tornam $\neg \phi'_i$ verdade, construir uma **forma normal disjuntiva** (DNF) para cada ϕ'_i

SAT $\leq_p$ 3-CNF-SAT

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow \neg y_3) \wedge (y_3 \leftrightarrow (x_1 \vee \neg x_2))$$

y_1	x_1	y_2	$y_1 \leftrightarrow (x_1 \wedge y_2)$
V	V	V	V
V	V	F	F
V	F	V	F
V	F	F	F
F	V	V	F
F	V	F	V
F	F	V	V
F	F	F	V

Disjuntiva

$$\neg\phi_2'' = (y_1 \wedge x_1 \wedge \neg y_2) \vee (y_1 \wedge \neg x_1 \wedge y_2) \vee (y_1 \wedge \neg x_1 \wedge \neg y_2) \vee (\neg y_1 \wedge x_1 \wedge y_2)$$

Cada cláusula de ϕ' introduz no máximo **8 cláusulas** em ϕ'', pois cada cláusula de ϕ' possui no máximo **3 variáveis**.

UDESC

$SAT \leq_p$ 3-CNF-SAT

Disjuntiva

$$\neg\phi_2'' = (y_1 \wedge x_1 \wedge \neg y_2) \vee (y_1 \wedge \neg x_1 \wedge y_2) \vee (y_1 \wedge \neg x_1 \wedge \neg y_2) \vee (\neg y_1 \wedge x_1 \wedge y_2)$$

Converter a fórmula para a CNF usando as **leis de De Morgan** (“inversão da disjuntiva”):

Conjuntiva

$$\phi_2'' = (\neg y_1 \vee \neg x_1 \vee y_2) \wedge (\neg y_1 \vee x_1 \vee \neg y_2) \wedge (\neg y_1 \vee x_1 \vee y_2) \wedge (y_1 \vee \neg x_1 \vee \neg y_2)$$

UDESC

SAT $\leq_p$ 3-CNF-SAT

O último passo faz com que cada cláusula tenha exatamente 3 literais, para isso usamos duas novas variáveis p e q. Para cada cláusula C_i em ϕ'' :

1. Se C_i tem 3 literais, simplesmente inclua C_i.
2. Se C_i tem 2 literais, $C_i = (l_1 \vee l_2)$, inclua:

$$(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$$

3. Se C_i tem 1 literal, l_1, inclua:

$$(l_1 \vee p \vee q) \wedge (l_1 \vee \neg p \vee \neg q) \wedge (l_1 \vee p \vee \neg q) \wedge (l_1 \vee \neg p \vee q)$$

Introduz no máximo **4** cláusulas por cláusula em ϕ''.

UDESC

$SAT \leq_p$ 3-CNF-SAT

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow \neg y_3) \wedge (y_3 \leftrightarrow (x_1 \vee \neg x_2))$$

$$\phi_1''' = (y_1 \vee p \vee q) \wedge (y_1 \vee \neg p \vee \neg q) \wedge (y_1 \vee p \vee \neg q) \wedge (y_1 \vee \neg p \vee q)$$

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow \neg y_3) \wedge (y_3 \leftrightarrow (x_1 \vee \neg x_2))$$

$$(y_1 \vee p \vee q) \wedge (y_1 \vee \neg p \vee \neg q) \wedge (y_1 \vee p \vee \neg q) \wedge (y_1 \vee \neg p \vee q) \wedge$$

$$(\neg y_1 \vee \neg x_1 \vee y_2) \wedge (\neg y_1 \vee x_1 \vee \neg y_2) \wedge (\neg y_1 \vee x_1 \vee y_2) \wedge (y_1 \vee \neg x_1 \vee \neg y_2)$$

3-CNF-SAT

Problema: verificar se uma fórmula booleana na 3-CNF é satisfazível.

3-CNF-SAT é *NP-Completo*? SIM

- **Passo 1 (Decisão polinomial)**: 3-CNF-SAT $\in$ *NP*. (sim)

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$$

- **Passo 2 (Redução de SAT)**: SAT $\leq_p$ 3-CNF-SAT. (sim)

$$\phi = x_1 \wedge \neg(x_1 \vee \neg x_2)$$ SAT

$$(y_1 \vee p \vee q) \wedge (y_1 \vee \neg p \vee \neg q) \wedge (y_1 \vee p \vee \neg q) \wedge (y_1 \vee \neg p \vee q) \wedge$$
$$(\neg y_1 \vee \neg x_1 \vee y_2) \wedge (\neg y_1 \vee x_1 \vee \neg y_2) \wedge (\neg y_1 \vee x_1 \vee y_2) \wedge (y_1 \vee \neg x_1 \vee \neg y_2) \wedge \ldots$$

Reduções

Resumindo, quais reduções de problemas foram feitas: