UDESC

Tratabilidade

Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

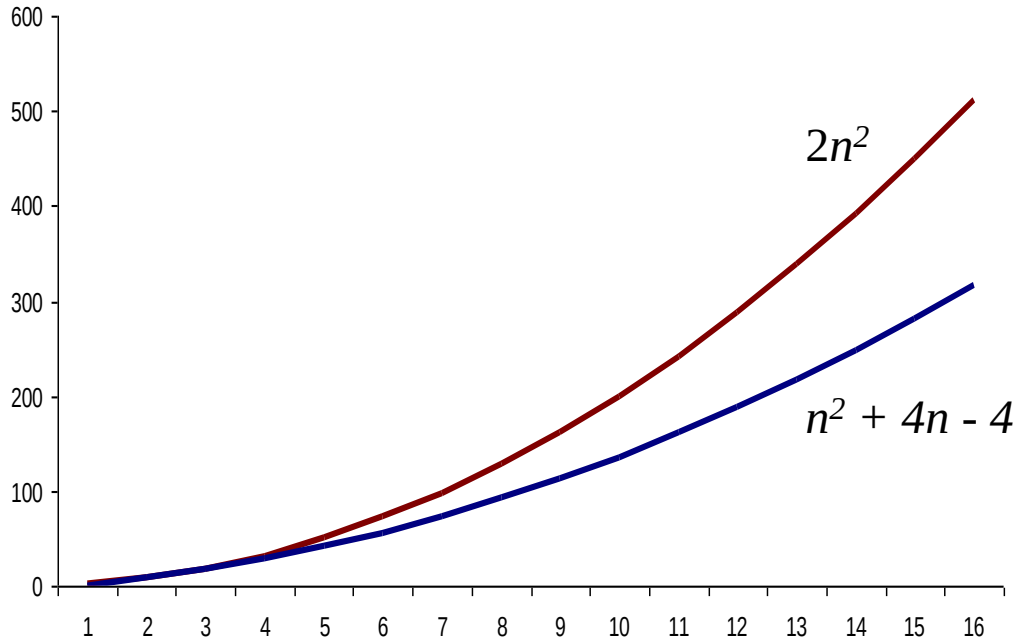
UDESC

Crescimento de Funções

Para simplificar o processo usamos uma abstração que ignora o custo de cada instrução, que é constante. Concentrando a análise no crescimento do tempo de execução (ou de outro recurso) em relação ao crescimento da entrada.

Notação Assintótica
(Notação O grande – Limite Superior)

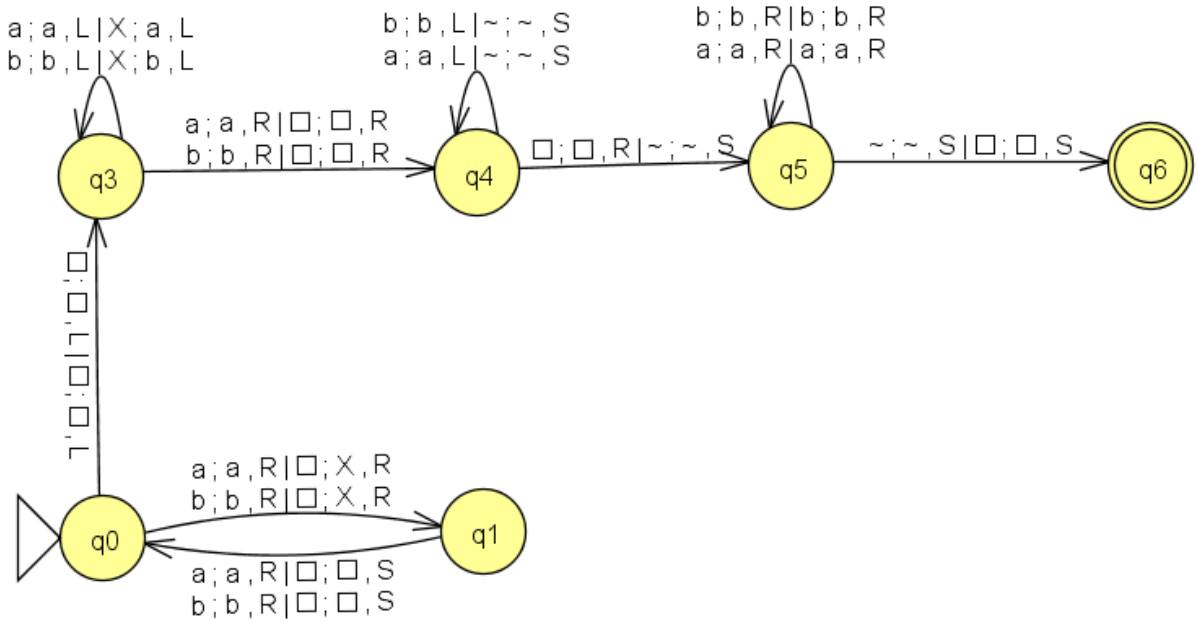
Uma função *f(n)* domina assintoticamente outra função *g(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|g(n)| \leq c.|f(n)|$. $g(n) = O(f(n))$



$$n^2 + 4n - 4 = O(n^2)$$

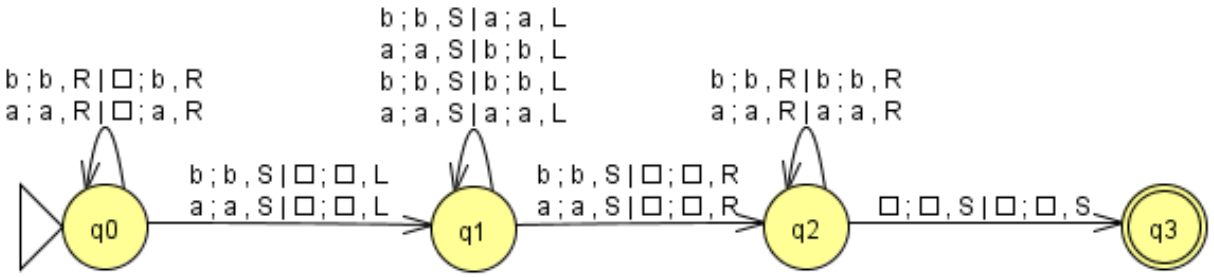
Notação Assintótica (Notação O grande – Limite Superior)

Para definir a complexidade de tempo de uma máquina de Turing consideramos o número de passos (transições) executados para reconhecer a entrada.



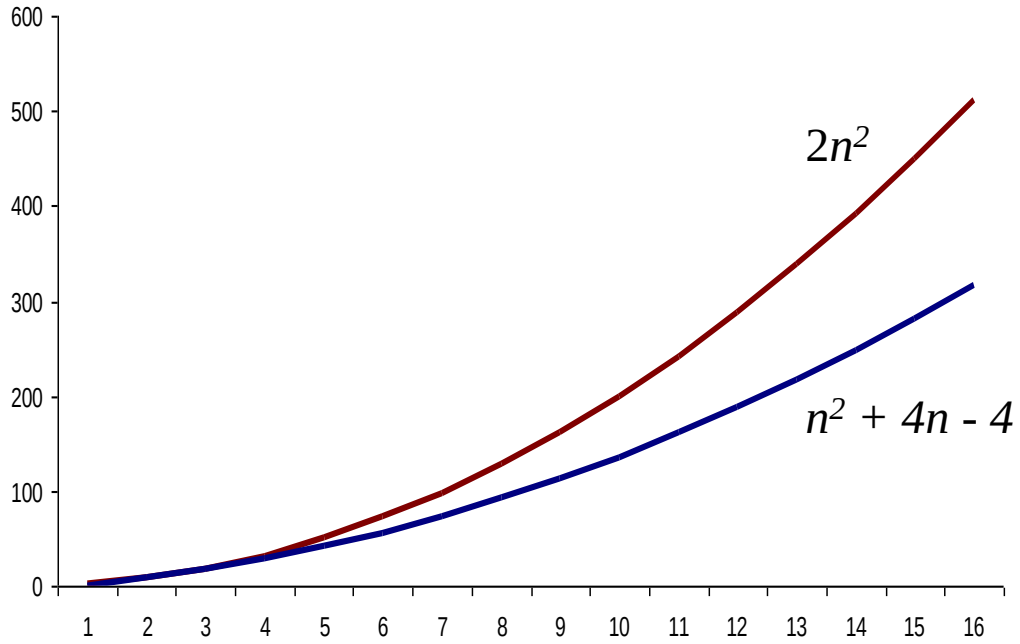
Notação Assintótica
(Notação O grande – Limite Superior)

Para definir a complexidade de tempo de uma máquina de Turing não-deterministas consideramos o número de passos (transições) executados para reconhecer a entrada na sequência de transições que leva a aceitação.



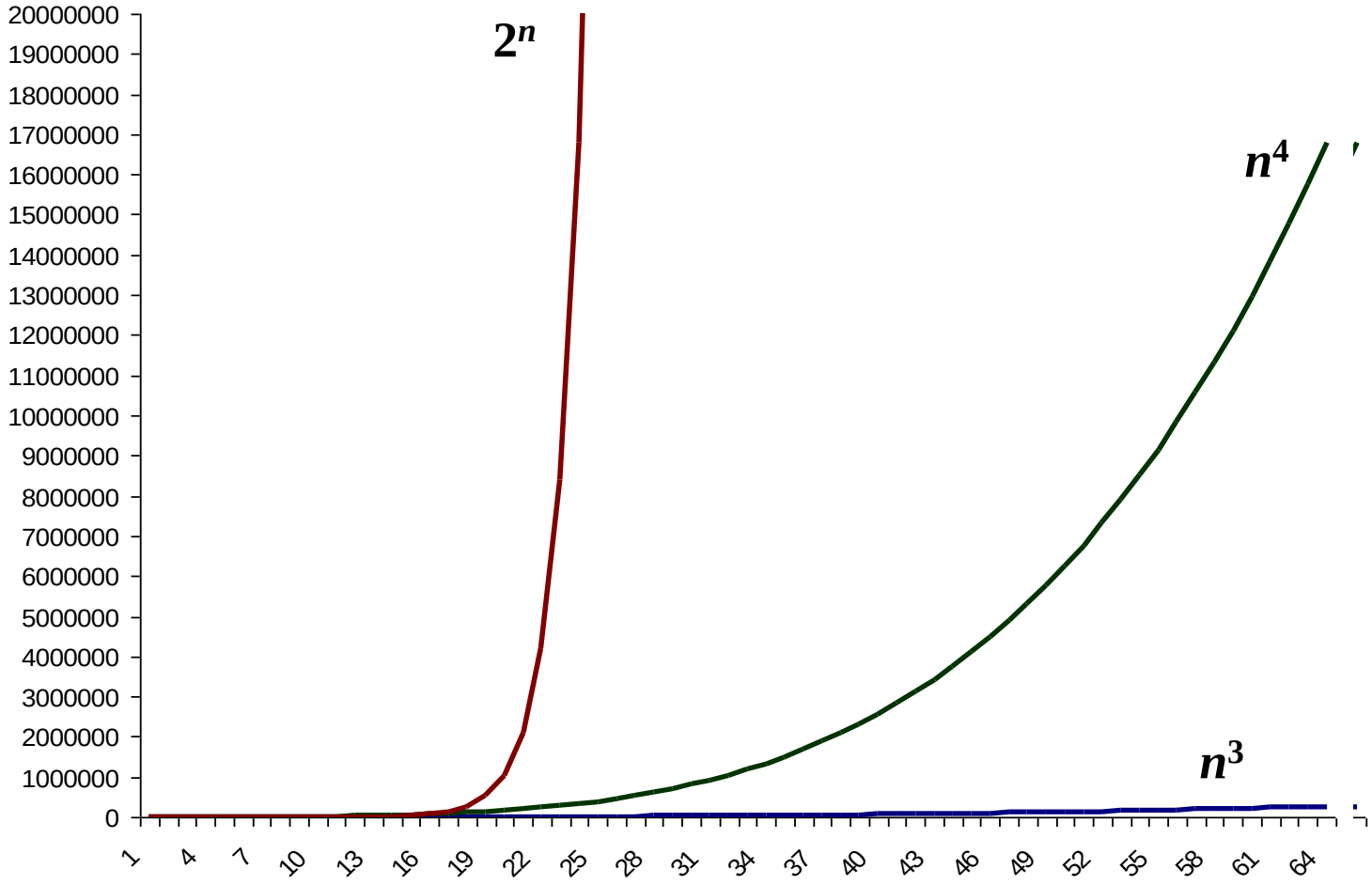
Notação Assintótica
(Notação O grande – Limite Superior)

Uma função *f(n)* domina assintoticamente outra função *g(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|g(n)| \leq c.|f(n)|$. $g(n) = O(f(n))$



$$n^2 + 4n - 4 = O(n^2)$$

Crescimento de Funções



Hierarquia de funções

Hierarquia de funções do ponto de vista assintótico:

$$1 \prec \log\log n \prec \log n \prec n^{\varepsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n} \prec c^{c^{n}}$$

onde ε e c são constantes arbitrárias tais que $0 < \varepsilon < 1 < c$.

Problemas tratáveis e intratáveis

Problemas tratáveis: resolvidos por algoritmos que executam em tempo polinomial.

Problemas intratáveis: não se conhece algoritmos que os resolvam em tempo polinomial.

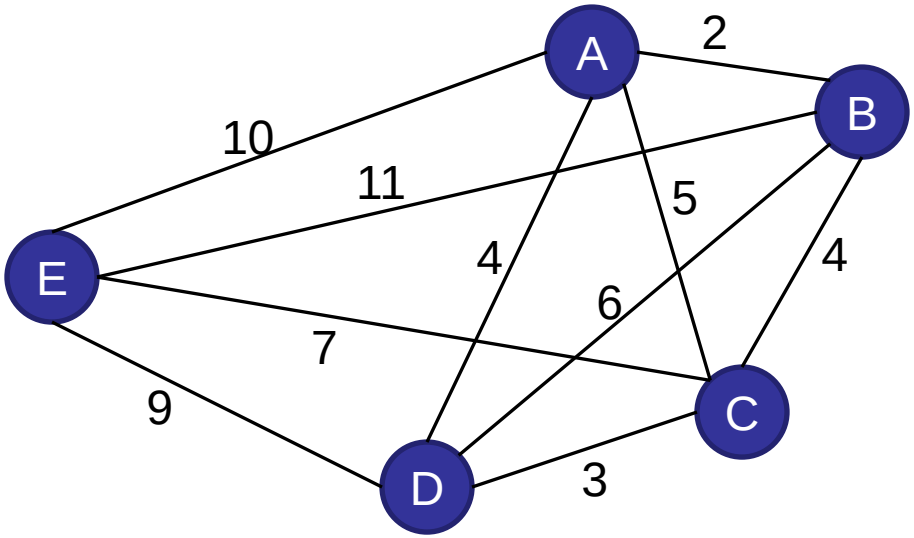
Problemas de Decisão.

Problemas de Otimização: Cada solução possível tem um valor associado e desejamos encontrar a solução com melhor valor.

Problemas de Decisão: Problemas que tem resposta sim ou não.

Problema do Caixeiro Viajante

Consistem em descobrir o caminho com custo mínimo para um vendedor que deseja visitar *n* cidades e retornar a cidade de origem. O problema é representado por um grafo não orientado completo ponderado com *n* vértices. Sendo que o valor *c*(*u*, *v*) (associado a cada aresta) representa o custo para viajar da cidade *u* a cidade *v*.



Problema do Caixeiro Viajante

Existe um caminho com custo menor que *k*?

Algoritmos Não Deterministas

Capaz de escolher uma entre várias alternativas possíveis a cada passo. A alternativa escolhida será sempre a alternativa que leva a conclusão esperada, caso essa alternativa exista.

```
int pesquisa(Estrutura *v, int n, int chave)
{
        int i;

        for (i = 0; i < n; i++)
                if (v[i].chave == chave)
                        return i;
        return -1;
}
```

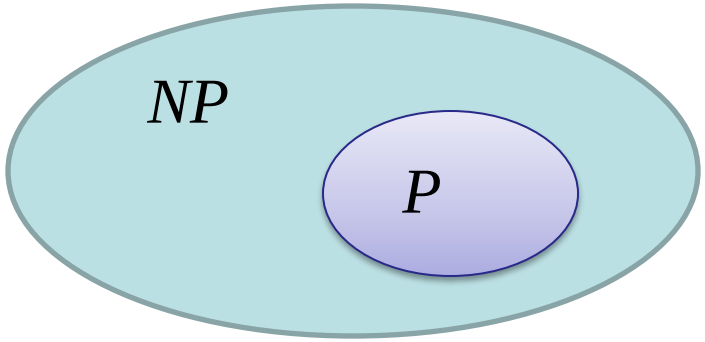
```
int pesquisa(Estrutura *v, int n, int chave)
{
        int i;

        i = escolheND(0, n - 1);
        if (v[i].chave == chave)
                return i;
        return -1;
}
```

Classes de Problemas *P* e *NP*

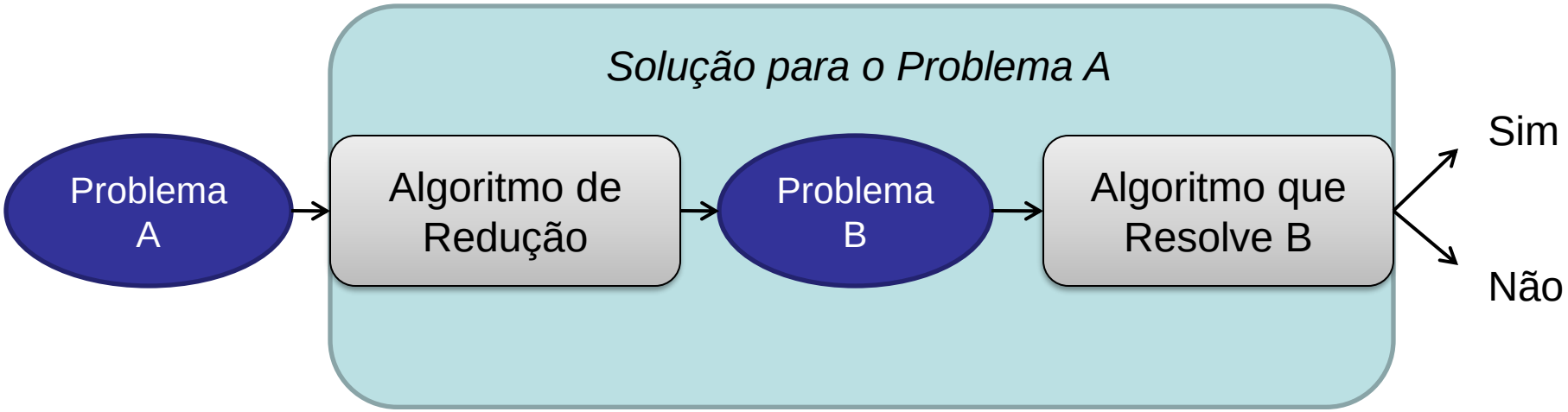
Classe de Problemas *P*: Problemas que podem ser resolvido (por algoritmos deterministas) em tempo polinomial.

Classe de Problemas *NP*: Problemas que podem ser resolvidos por algoritmos não deterministas em tempo polinomial. Ou problemas que a solução pode ser verificada em tempo polinomial.

Possível relação entre as classes:



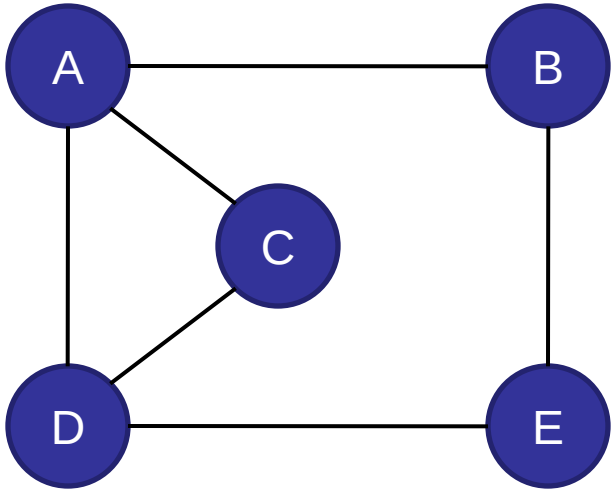
Redução de Problemas



Ciclo Hamiltoniano

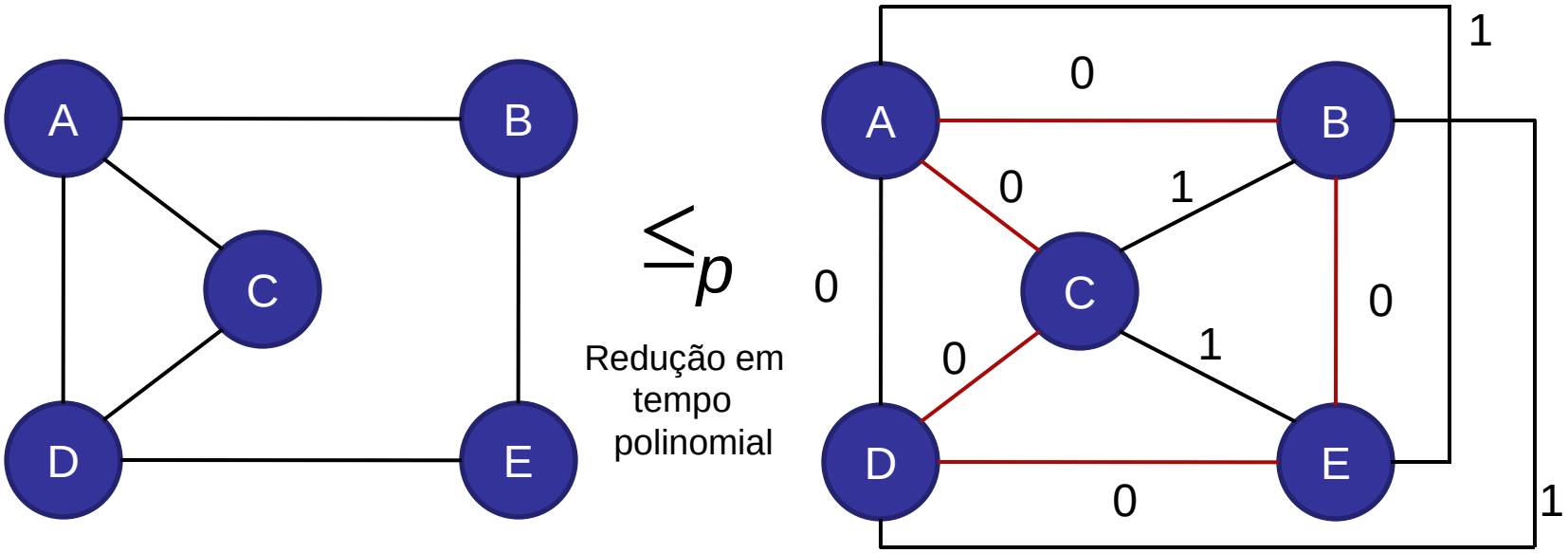
Um ***Ciclo Hamiltoniano*** em um grafo não orientado é um caminho que passa por cada vértice do grafo exatamente uma vez.

Problema do ***Ciclo Hamiltoniano***: Um grafo *G* possui um ciclo Hamiltoniano?



Redução do Problema do Ciclo Hamiltoniano ao Problema do Caixeiro Viajante

G $(G', 0)$



para cada vértice i

para cada vértice j

se $(i, j) \in H$ então $c(i, j) \leftarrow 0$

senão $c(i, j) \leftarrow 1$

(SAT) Satisfazibilidade de Fórmulas Booleanas

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

Exemplo:

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

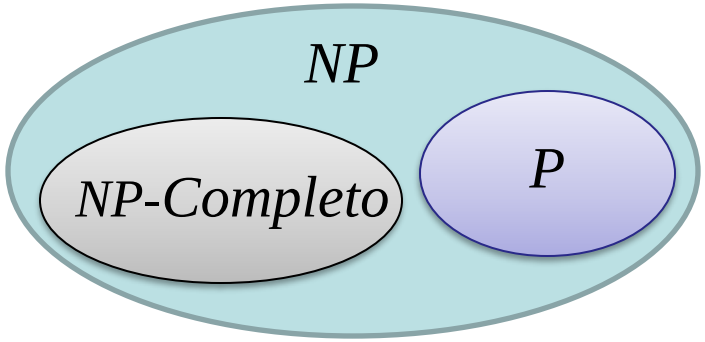
NP-Completo

Teorema de Cook: SAT está em P se e somente se $P = NP$.

Um problema X é ***NP-Completo*** se:

1. $X \in NP$
2. $X' \leq_p X$ para todo $X' \in NP$.

Possível relação entre as classes:



UDESC

Forma Normal Conjuntiva

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

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:

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

UDESC

3-CNF-SAT

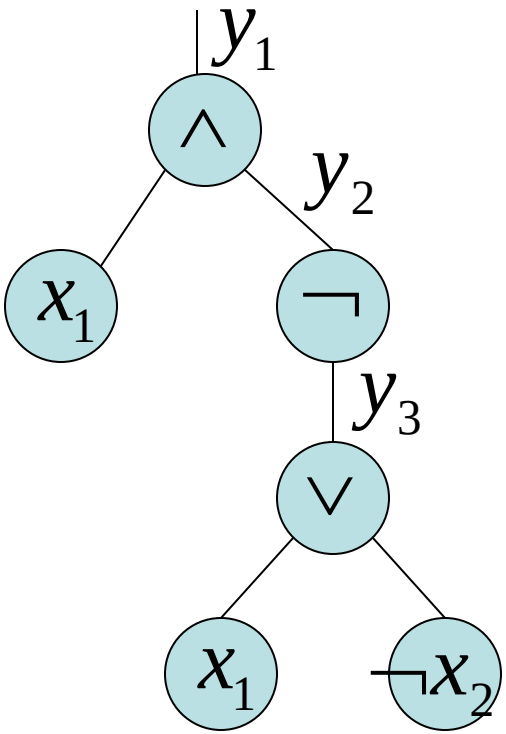
Verificar se uma fórmula booleana na 3-CNF é satisfazível.

3-CNF-SAT é *NP-Completo*:

- 3-CNF-SAT $\in$ *NP*.
- SAT $\leq_p$ 3-CNF-SAT.

$SAT \leq_p$ 3-CNF-SAT

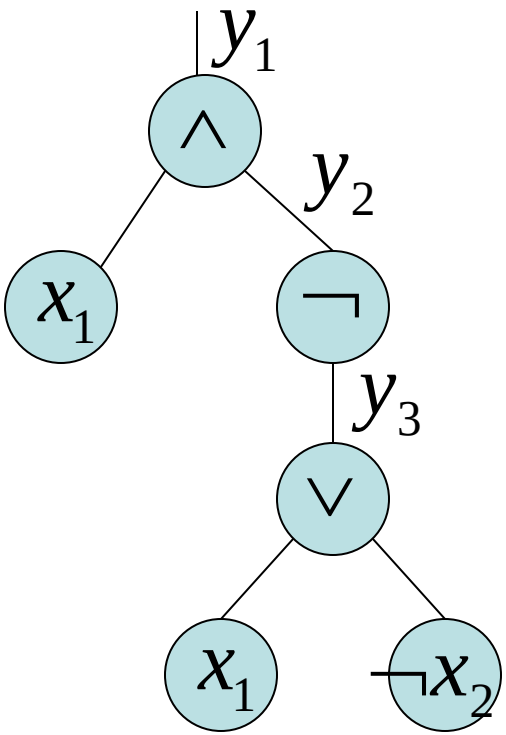
Dada uma fórmula booleana: $\phi = x_1 \wedge \neg(x_1 \vee \neg x_2)$



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 cláusulas que descrevem as operações de cada nó.

Introduz 1 variável e 1 cláusula para cada operador.

UDESC

$SAT \leq_p$ 3-CNF-SAT

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

Para cada ϕ'_i construir uma tabela tabela verdade, usando as entradas que tornam $\neg \phi'_i$ verdade, construir uma forma normal disjuntiva 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	$\phi'_2 = y_1 \leftrightarrow (x_1 \wedge y_2)$
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

$$\neg\phi''_2 = (y_1 \wedge x_1 \wedge \neg y_2)$$
$$\vee (y_1 \wedge \neg x_1 \wedge y_2)$$
$$\vee (y_1 \wedge \neg x_1 \wedge \neg y_2)$$
$$\vee (\neg y_1 \wedge x_1 \wedge y_2)$$

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

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

Converter a fórmula para a CNF usando as leis de De Morgan:

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

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

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

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

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

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

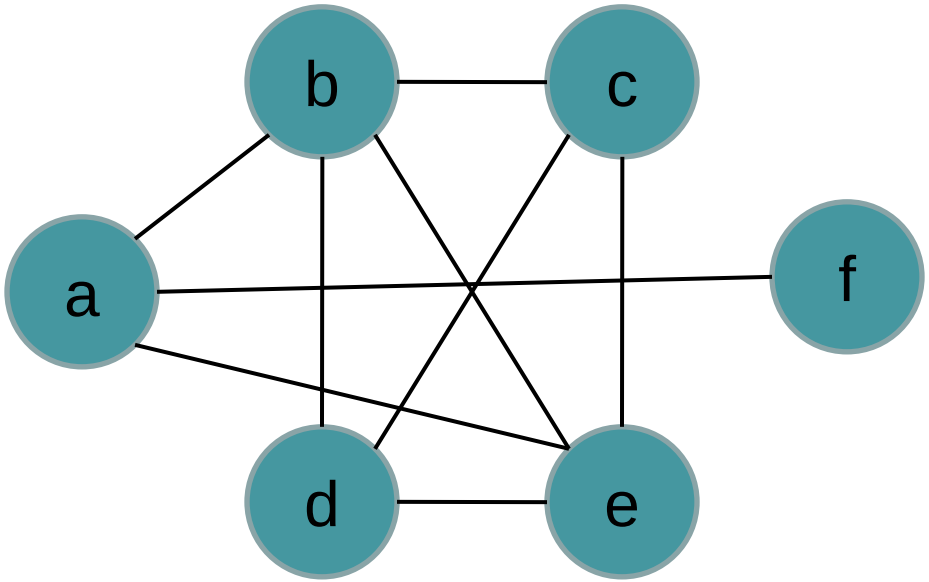
$$(\neg y_1 \vee \neg x_1 \vee y_2) \wedge (\neg y_1 \vee x_1 \vee \neg y_2) \wedge (\neg y_1 \vee x_1 \vee y_2) \wedge (y_1 \vee \neg x_1 \vee \neg y_2)$$

CLIQUE

Um *Clique* em um grafo não direcionado $G = (V, A)$ é um subconjunto de vértices $V' \subseteq V$, onde cada vértice está conectado por uma aresta. Ou seja um subgrafo completo.

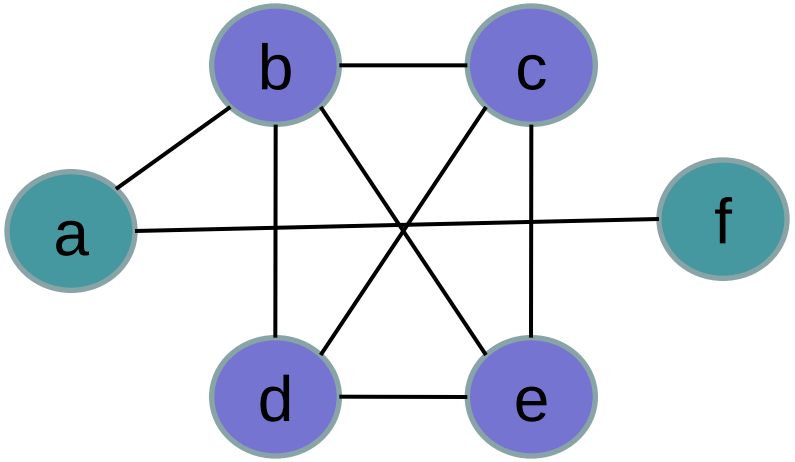
Versão de otimização: Encontrar o maior *Clique* possível.

Versão de decisão: Existe um *Clique* de tamanho $\geq k$?



CLIQUE

Clique $\in NP$



Dado um grafo $G = (V, A)$, a solução (certificado) V' e k

Verificar de $|V| \geq k$

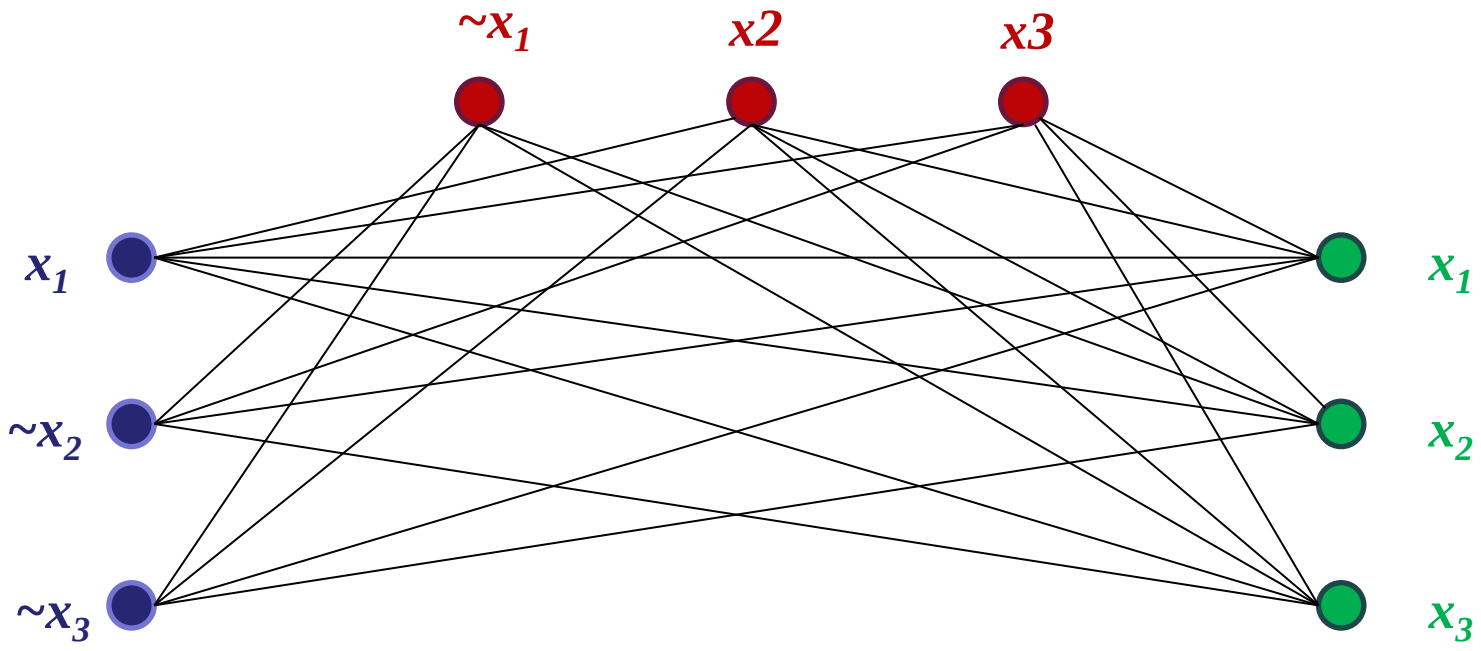
Para cada $u \in V'$

    Para cada $v \in V'$

        Se $u \neq v$ então verificar se $(u, v) \in A$

3-CNF-SAT $\leq_p$ CLIQUE

$\phi = (x_1 \vee \sim x_2 \vee \sim x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



Existe um clique de tamanho k? Sendo k o número de cláusulas.

UDESC

3-CNF-SAT $\leq_p$ CLIQUE

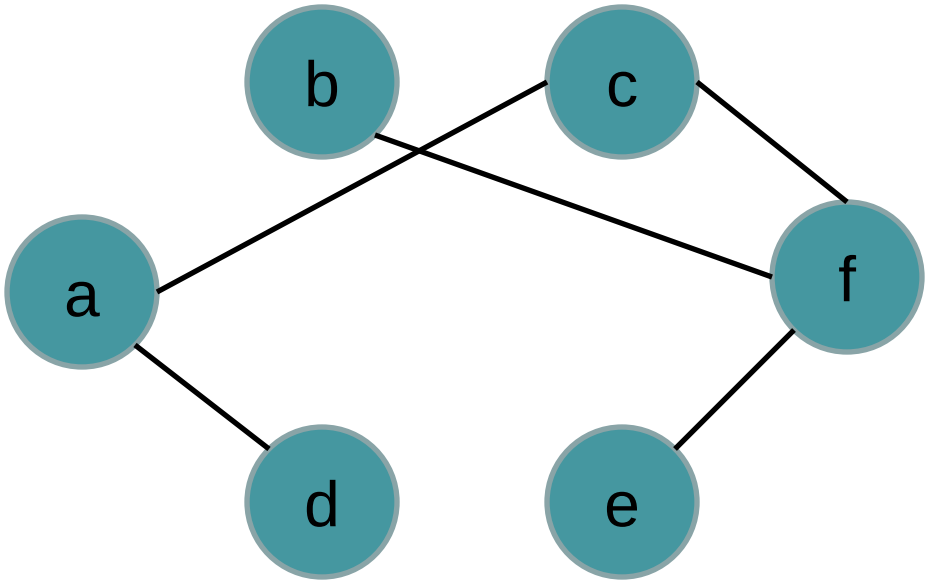
Dada uma instancia ϕ do problema 3-CNF-SAT, cada cláusula de ϕ gera 3 vértices, sendo que cada vértice corresponde a um literal da cláusula.

É adicionada uma aresta para cada par de vértices ***u*** e ***v*** se as duas condições a seguir forem satisfeitas:

- Se ***u*** e ***v*** não foram gerados a partir da mesma cláusula;
- Se ***u*** e ***v*** não foram gerados a partir de um literal que corresponde a uma variável e sua negação. Por exemplo, um vértice correspondente a variável $\boldsymbol{x_1}$ não pode ser conectado a um vértice correspondente a negação de $\boldsymbol{x_1}$.

Cobertura de Vértices
(VERTEX-COVER)

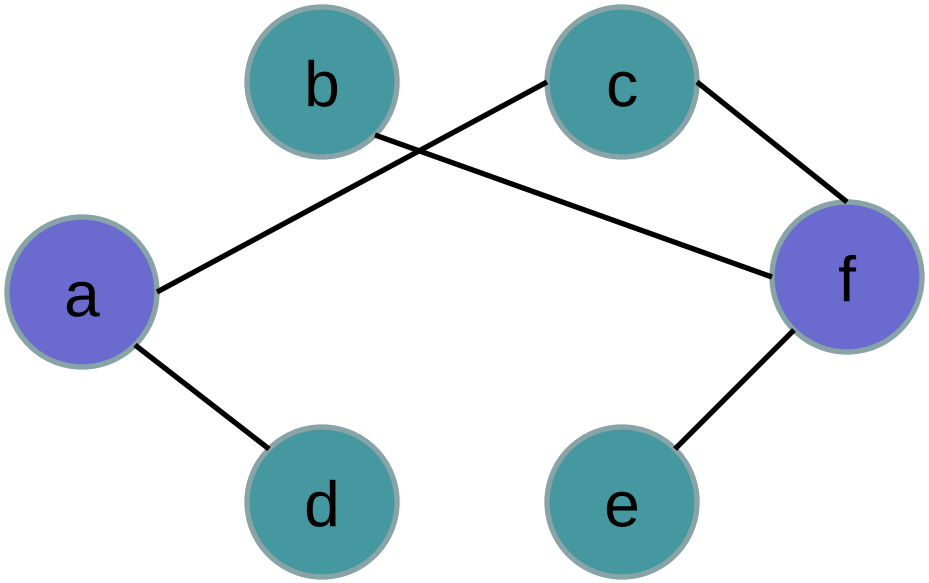
Uma *Cobertura de Vértices* de um grafo não orientado $G = (V, A)$ é um subconjunto $V' \subseteq V$ tal que se $(u, v) \in A$, então $u \in V'$ ou $v \in V'$.



Cobertura de Vértices
(VERTEX-COVER)

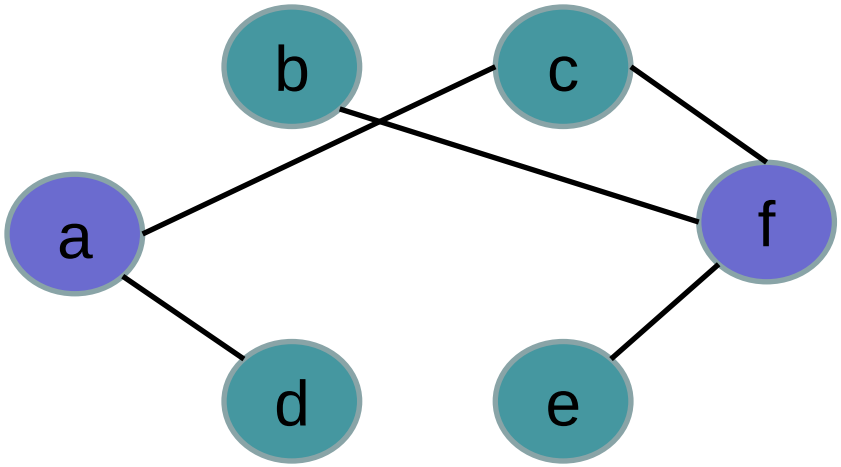
Versão de otimização: Encontrar menor Cobertura de Vértices.

Versão de decisão: Existe uma cobertura de tamanho *k*?



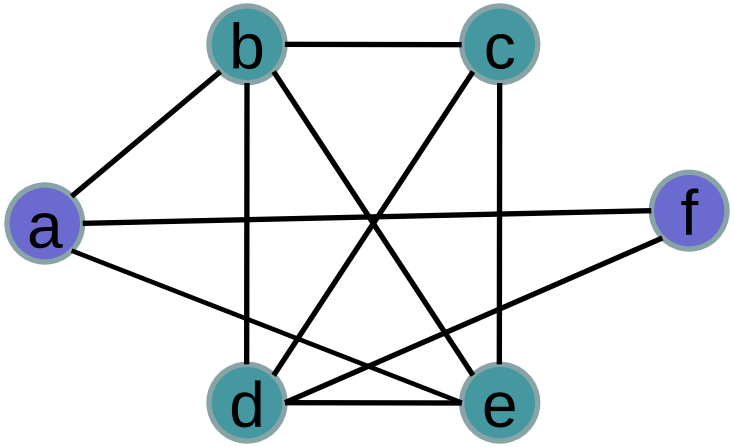
Cobertura de Vértices
(VERTEX-COVER)

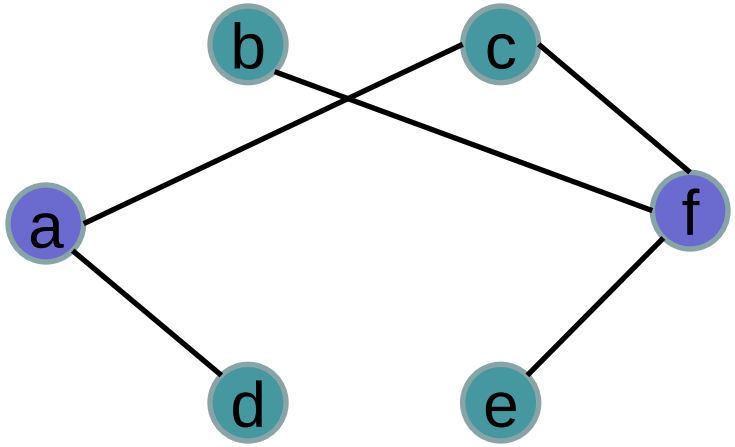
Cobertura de Vértices $\in$ *NP*.



Dado um grafo $G = (V, A)$ e a solução (certificado) V'
Verificar de $|V| \geq k$
Para cada $(u, v) \in A$
    Verificar se $u \in V'$ ou $v \in V'$

CLIQUE $\leq_p$ VERTEX-COVER





CLIQUE
Entrada (G, k), onde $G = (V, A)$

VERTEX-COVER
Entrada $(\overline{G}, |V| - k)$

UDESC

SUBSET-SUM

Dado um conjunto finito de inteiros positivos S e um inteiro $t > 0$, determinar se existe um subconjunto $S' \subseteq S$ onde o somatório dos elementos de S' é igual a t.

$$\sum_{i=1}^{n} s'_i = t$$

3-CNF-SAT $\leq_p$ SUBSET-SUM

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

	x_1	x_2	x_3	C_1	C_2
v_1	1	0	0	0	1
v'_1	1	0	0	1	0
v_2	0	1	0	1	1
v'_2	0	1	0	0	0
v_3	0	0	1	0	0
v'_3	0	0	1	1	1
s_1	0	0	0	1	0
s'_1	0	0	0	2	0
s_2	0	0	0	0	1
s'_2	0	0	0	0	2
t	1	1	1	4	4