UDESC

Complexidade de Algoritmos

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

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

UDESC

Reduções de Problemas

X ∈ 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 $Y \in NP$

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 $\in$ NP-Completo

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.

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

3-CNF-SAT ∈ NP-Completo

Passo 1: 3-CNF-SAT ∈ *NP*.

```
1. Para cada símbolo y da entrada w:
    i. Simule a operação booleana sempre que
       possível, armazenando os resultados parciais
       ou os símbolos que ainda não puderam ser
       simplificados
2. Considerando uma fórmula booleana válida,
   retorne o último valor booleano restante como
   resposta (V = aceite / F = rejeite)
```

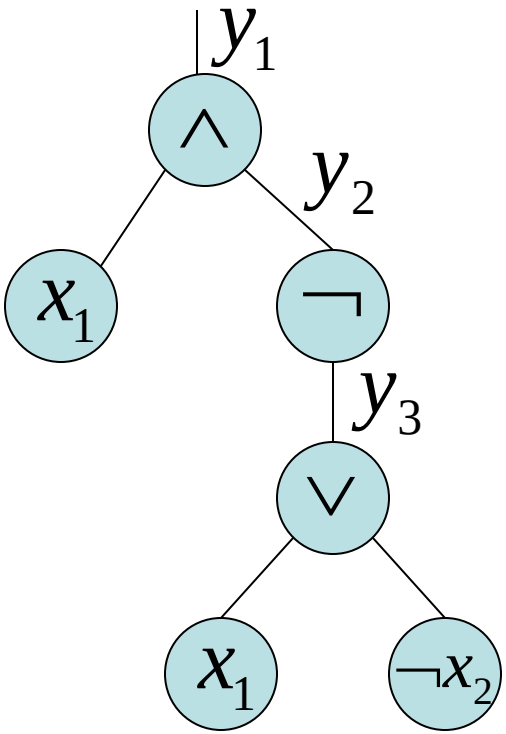
$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$$

Neste caso qual seria a complexidade do algoritmo?

3-CNF-SAT $\in$ NP-Completo

Passo 2: SAT $\leq_p$ 3-CNF-SAT.

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



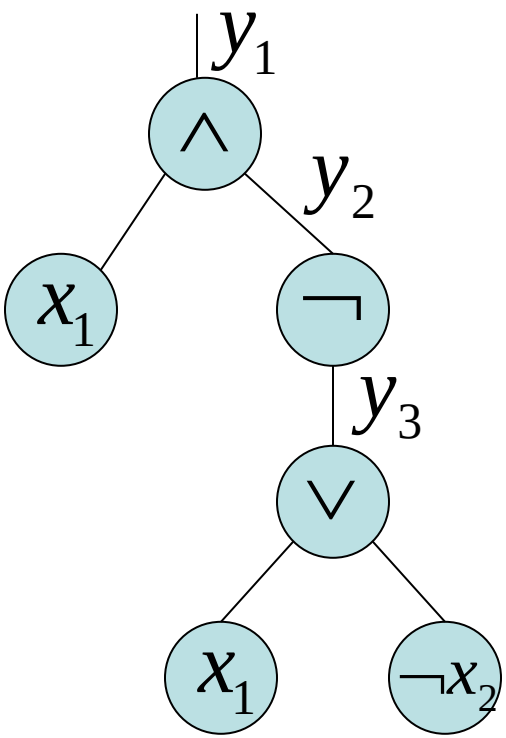
SAT

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.

3-CNF-SAT $\in$ NP-Completo

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



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.

3-CNF-SAT $\in$ NP-Completo

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

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

3-CNF-SAT ∈ NP-Completo

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

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

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

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

UDESC

3-CNF-SAT $\in$ NP-Completo

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

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

3-CNF-SAT $\in$ NP-Completo

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''.

3-CNF-SAT $\in$ NP-Completo

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

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

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

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

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

3-CNF-SAT $\in$ NP-Completo

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

3-CNF-SAT é *NP-Completo*? SIM

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

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

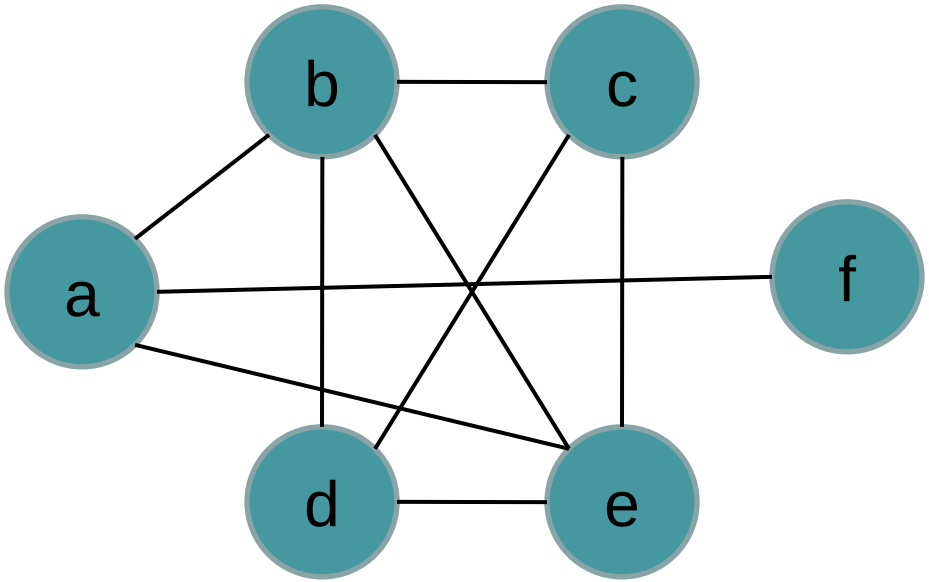
$$(y_1 \vee p \vee q) \wedge (y_1 \vee \neg p \vee \neg q) \wedge (y_1 \vee p \vee \neg q) \wedge (y_1 \vee \neg p \vee q) \wedge$$
$$(\neg y_1 \vee \neg x_1 \vee y_2) \wedge (\neg y_1 \vee x_1 \vee \neg y_2) \wedge (\neg y_1 \vee x_1 \vee y_2) \wedge (y_1 \vee \neg x_1 \vee \neg y_2) \wedge \ldots$$

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.

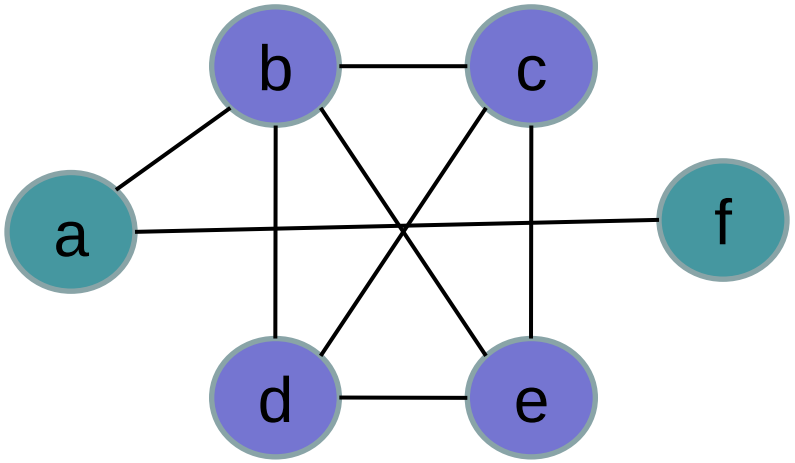
Versão de decisão: Existe um *Clique* de tamanho $\geq k$?



CLIQUE $\in$ NP-Completo

CLIQUE é *NP-Completo*?

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



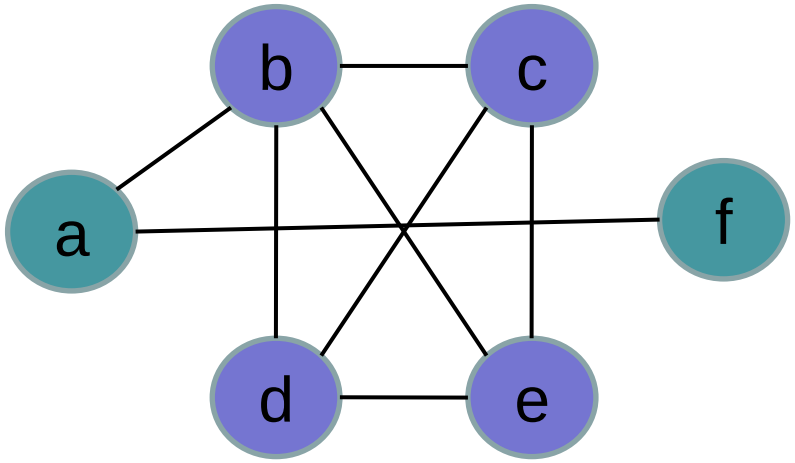
CLIQUE $\in$ NP-Completo

Passo 1: Clique $\in$ *NP*

V = { a, b, c, d, e, f }

A = { (a,b), (a,f), (b,c), (b,d), (b,e),
(c,d), (c,e), (d,e) }

V' = { b, c, d, e }



Dado um grafo $G = (V, A)$, a solução (**certificado**) V' e k, verificar se V' é válido e se $|V'| \geq k$ em tempo polinomial

Se $|V'| < k$ **então retorne** *Falso*
Para cada $u \in V'$
Para cada $v \in V'$
Se $u \neq v$ **então** verificar se $(u, v) \in A$

Complexidade?

CLIQUE $\in$ NP-Completo

- **Passo 2**: 3-CNF-SAT $\leq_p$ CLIQUE.

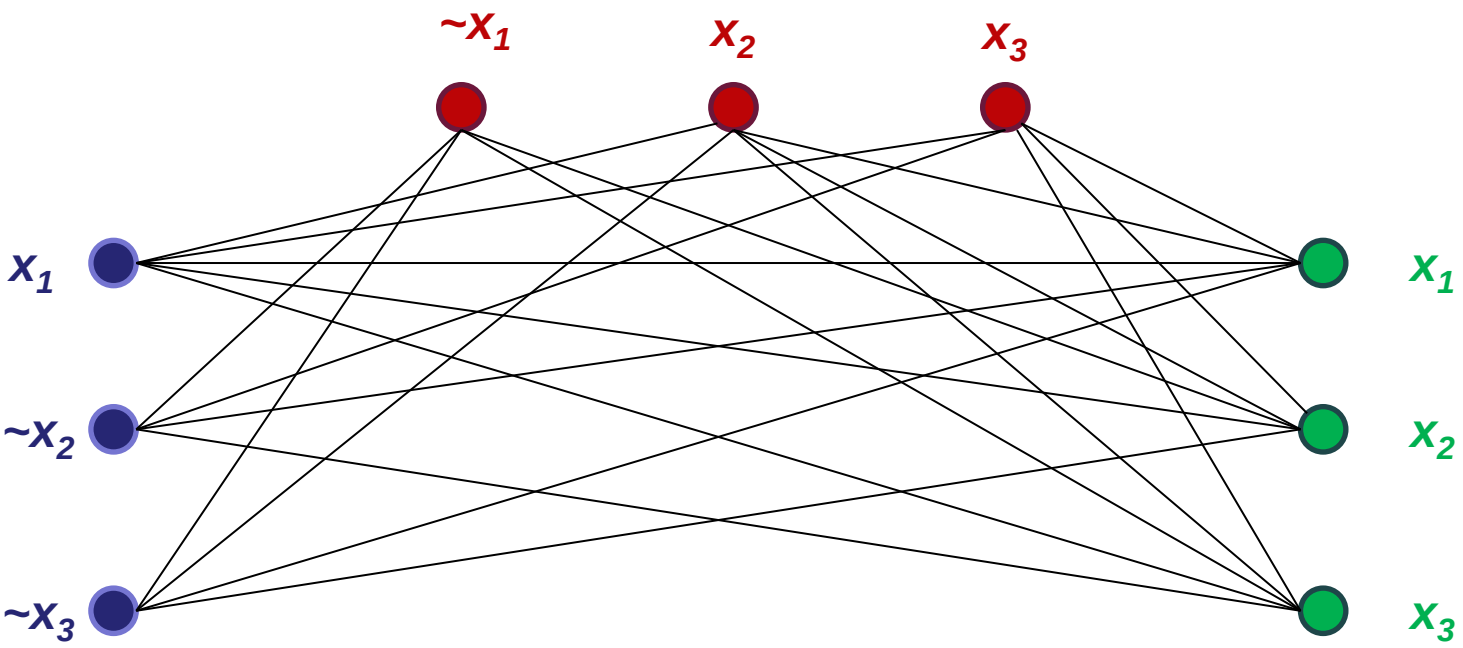
Dada uma instancia ϕ do problema 3-CNF-SAT converteremos esta para um grafo G que terá 3k vértices, onde k é o número de cláusulas de ϕ.

– ***u*** e ***v*** são vértices que correspondem a literais em diferentes cláusulas;
– Todos os vértices são ligados por arestas, com exceção:
  - se ***u*** e ***v*** pertencem a mesma cláusula, então não há ligação;
  - se ***u*** corresponde a um literal ***x***, e ***v*** corresponde ao literal ***~x,*** então não há ligação entre esses dois vértices;

CLIQUE ∈ NP-Completo

- **Passo 2**: 3-CNF-SAT $\leq_p$ CLIQUE.

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

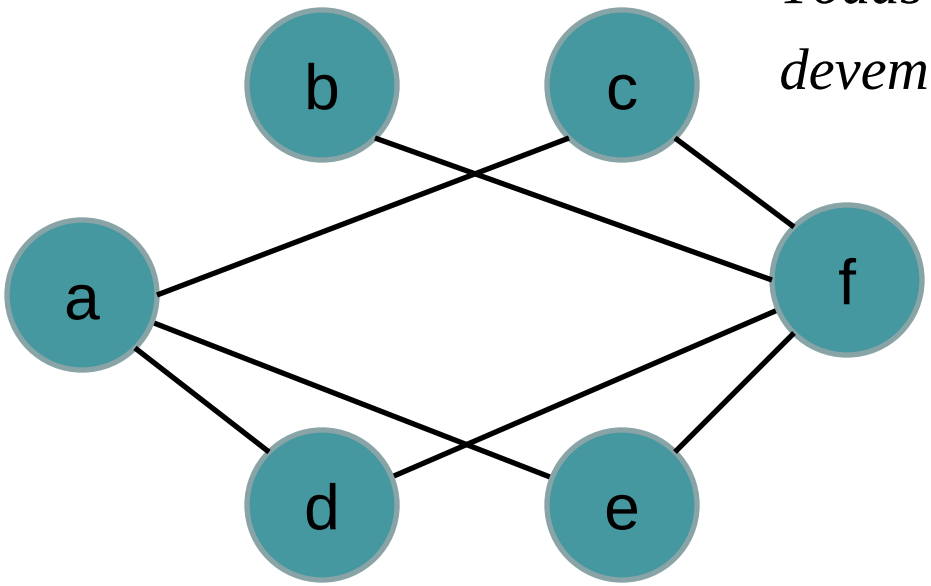


ϕ é satisfazível ↔ G possui um clique ≥ k

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 $v \in V'$.

Todas as arestas devem ser observadas!

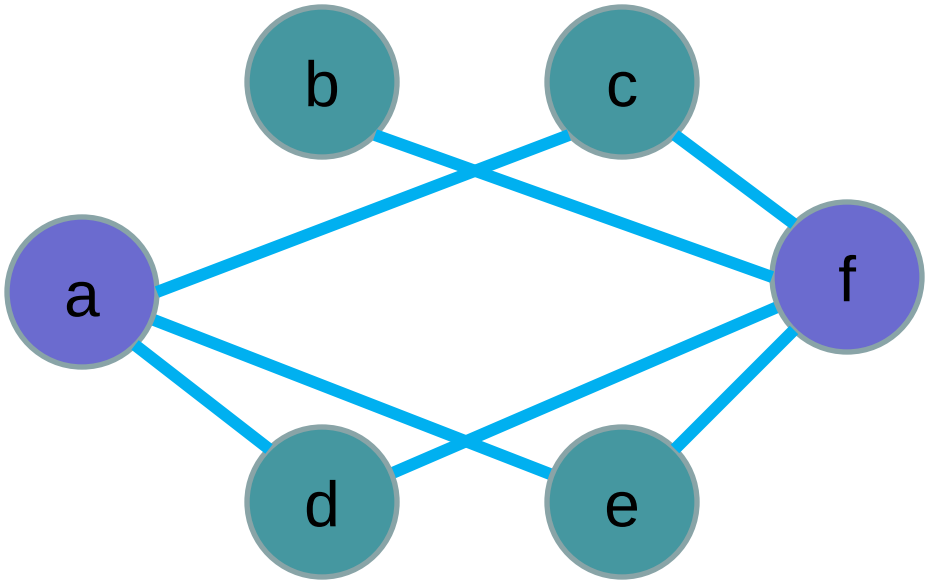


UDESC

Cobertura de Vértices
(VERTEX-COVER)

Versão de otimização: Encontrar menor Cobertura de Vértices.

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



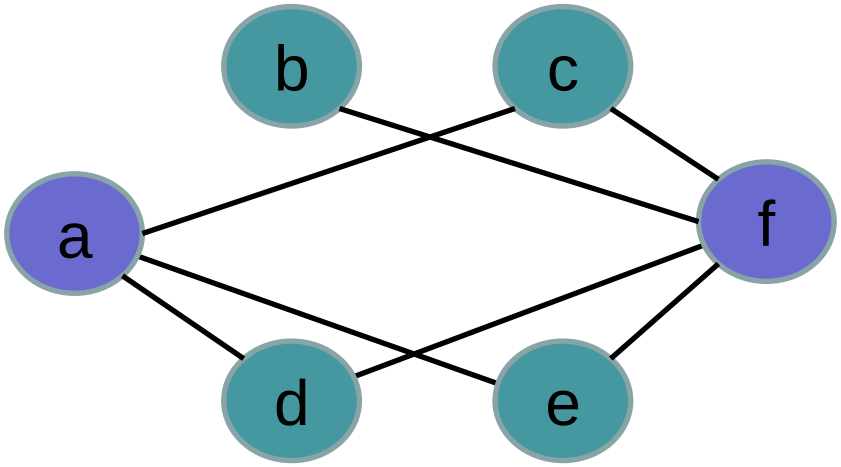
VERTEX-COVER ∈ NP-Completo

Passo 1: Cobertura de Vértices ∈ NP.

V = { a, b, c, d, e, f }

A = { (a,c), (a,d), (b,f), (c,f), (f,e) }

V' = { a, f }



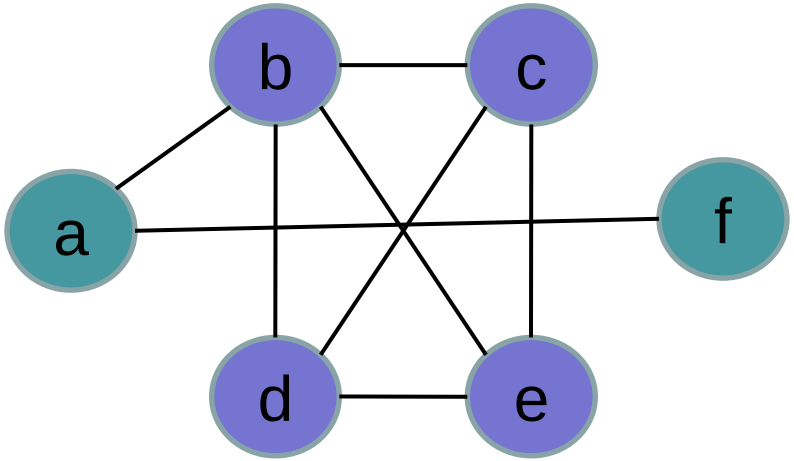
Dado um grafo $G = (V, A)$ e a solução (**certificado**) V' verificar se V' é válido e se $|V'| \leq k$ em tempo polinomial

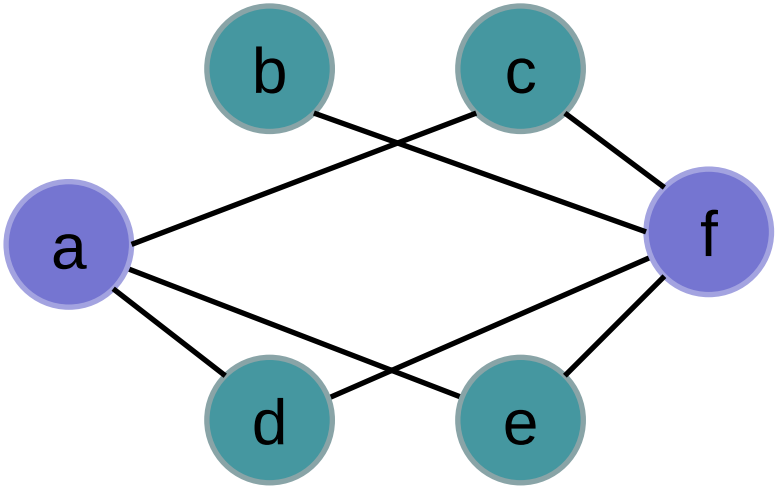
Se $|V'| > k$ **então retorne** *Falso*
Para cada $(u, v) \in A$
  Verificar se $u \in V'$ ou $v \in V'$

Complexidade?

VERTEX-COVER ∈ NP-Completo

- **Passo 2**: CLIQUE $\leq_p$ VERTEX-COVER





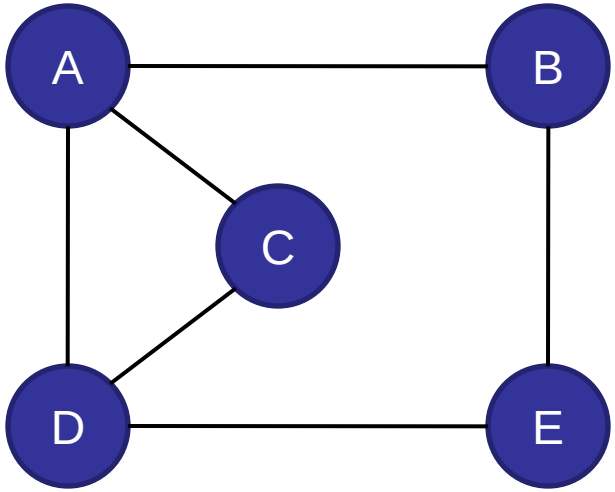
CLIQUE
Entrada (G, k), onde $G = (V, A)$

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

Ciclo Hamiltoniano

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

Versão de decisão: um grafo *G* possui um ciclo Hamiltoniano?



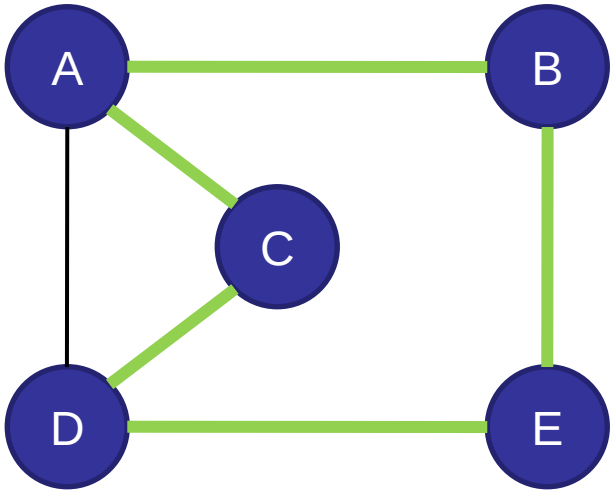
Ciclo Hamiltoniano ∈ NP-Completo

Passo 1: Ciclo Hamiltoniano ∈ *NP*

V = { a, b, c, d, e }

A = { (a,b), (a,c), (a,d), (b,e), (c,e), (d,e) }

V' = { a, b, e, d, c }



Dado um grafo $G = (V, A)$ e a solução (**certificado**) V'
verificar se V' é um ciclo Hamiltoniano em tempo polinomial

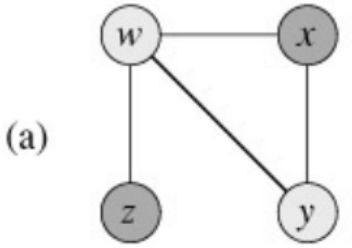
```
Para cada v ∈ V: viz[v] = não marcado
Para cada v' ∈ V':
     Se viz[v']==marcado: retorne falso
     Senão: viz[v'] = marcado
Para cada x ∈ viz:
     Se x não está marcado: retorne falso
retorne verdadeiro
```

Complexidade?

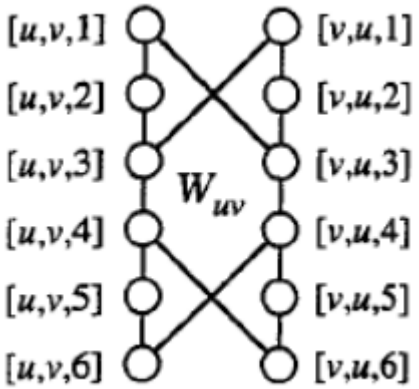
Vertex-Cover $\leq_p$ Ciclo Hamiltoniano

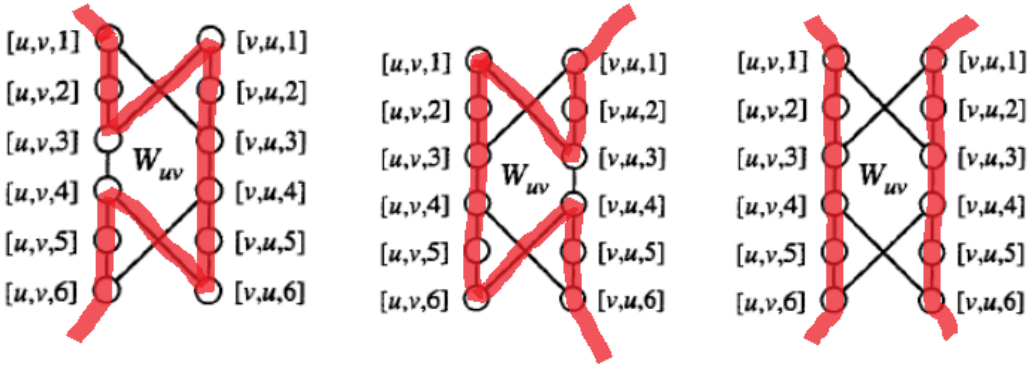
- **Passo 2**: VERTEX-COVER $\leq_p$ CICLO HAMILTONIANO

Dado um grafo instância do problema de Cobertura de vértices G = (V , E), devemos:



- criar ***k*** vértices seletores, onde ***k*** é o número de vértices que pertencem a solução da cobertura;
- criar ***E*** dispositivos, totalizando ***E* * 12** novos vértices e ***E* * 14*** arestas;





Ciclo Hamiltoniano ∈ NP-Completo

- **Passo 2**: VERTEX-COVER $\leq_p$ CICLO HAMILTONIANO

- criar uma lista com as adjacências de cada nó (para formar um caminho entre todas as coberturas de um vértices):

  u: $u_1, u_2, \ldots u_{grau(u)}$

  v: $v_1, v_2, \ldots v_{grau(v)}$

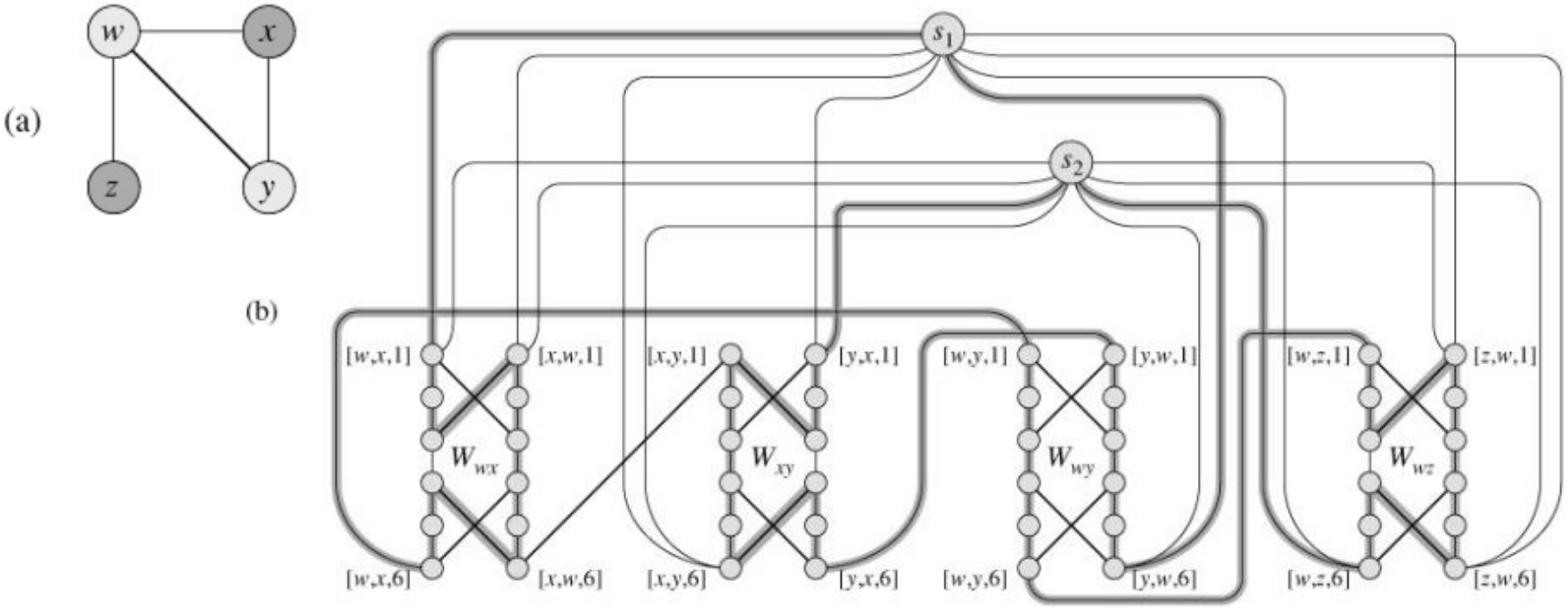
  - adicionar arestas para unir pares de dispositivos: $\{([u,u_i,6],[u,u_{i+1},1]), \ldots \}$
  - criar arestas para unir o primeiro $[u, u_1, 1]$ e o último vértice $[u, u_{grau(u)}, 6]$ de cada um desses caminhos a cada vértice seletor.

    $\{(sj, [u, u_1, 1]) : u \in V \text{ e } 1 \leq j \leq k\}$

    $\{(sj, [u, u_{grau(u)}, 6]) : u \in V \text{ e } 1 \leq j \leq k\}$

Ciclo Hamiltoniano ∈ NP-Completo

- **Passo 2**: VERTEX-COVER $\leq_p$ CICLO HAMILTONIANO



s1 → W_{wx} → W_{wy}* → W_{wz} → s2 → W_{yx} → W_{yw}* → s1

O caminho 3 entre dispositivos (*) só ocorre em arestas compartilhadas por vértices que fazem parte da solução da cobertura de vértices

Ciclo Hamiltoniano ∈ NP-Completo

- **Passo 2**: VERTEX-COVER $\leq_p$ CICLO HAMILTONIANO

Importante: note que o novo grafo G' = (V', E')

$| V' | = 12 |E| + k$

$| V' | \leq 12 |E| + |V|$

Instância cresceu apenas em tamanho polinomial

$| E' | = 14 |E| + (2|E| - |V|) + (2k |V|)$

$| E' | = 16 |E| + (2k – 1) |V|$

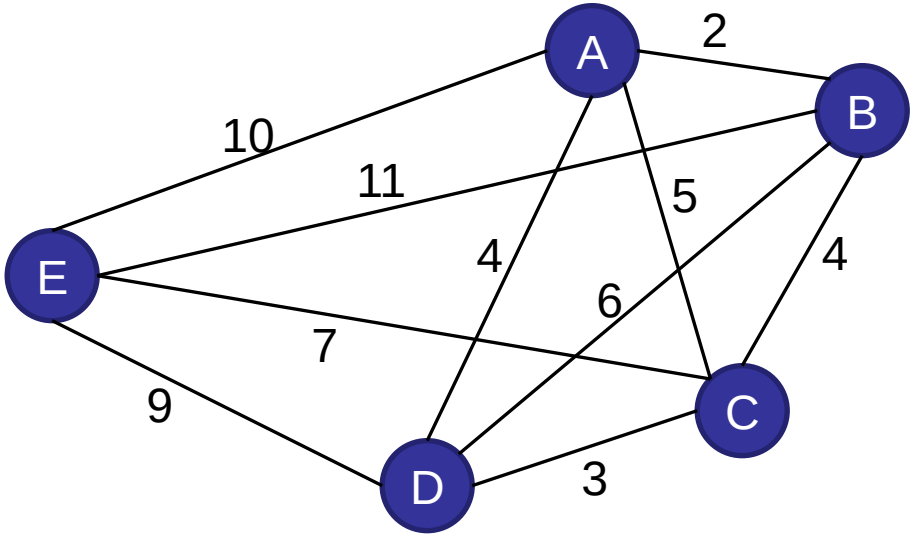
$| E' | \leq 16 |E| + (2|V| – 1) |V|$

Caixeiro Viajante

Um vendedor deseja visitar *n* cidades e retornar a cidade de origem. Dado um grafo não orientado completo com *n* vértices, onde existe um custo *c*(*i*, *j*) (associado a cada aresta) para viajar da cidade *i* a cidade *j*.

Otimização: Qual é o menor caminho para o vendedor?

Decisão: Existe um caminho para o vendedor com custo máximo igual a *k*?



Caixeiro Viajante ∈ NP-Completo

Passo 1: Caixeiro Viajante ∈ *NP*

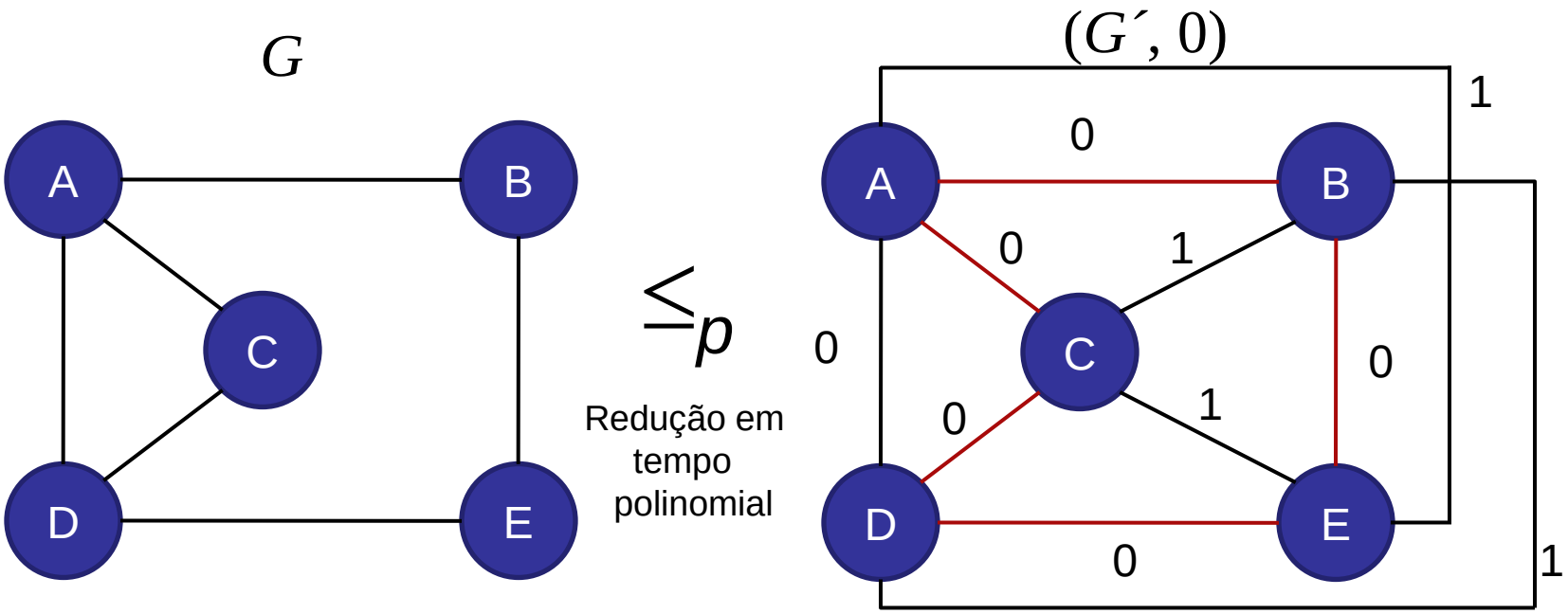
Dado um grafo $G = (V, A)$, a solução (**certificado**) V' e o custo máximo k, verificar se V' é um caminho válido do Caixeiro com custo menor ou igual a k em tempo polinomial

```
Para cada v ∈ V: viz[v] = não marcado
custo = 0, n = |V'|
Se V'[0] ≠ V'[n-1]: retorne falso
Para i=0 até n-2:
     Se viz[ V'[i] ]==marcado: retorne falso
     Senão: viz[ V'[i] ] = marcado
            custo = custo + G[ V'[i] ][ V'[i+1] ].custo
Se custo > k: retorne falso
Para i=0 até n-1:
     Se viz[ i ] == não marcado: retorne falso
retorne verdadeiro
```

Complexidade?

Caixeiro Viajante ∈ NP-Completo

- **Passo 2**: CICLO HAMILTON $\leq_p$ CAIXEIRO



para cada vértice i
  para cada vértice j
    se $(i, j) \in H$ então $c(i, j) \leftarrow 0$
    senão $c(i, j) \leftarrow 1$

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.

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

Exemplo: S = {1, 2, 7, 14, 49, 98, 343, 686, 2.409, 2.793, 16.808, 17.206, 17.705, 117.993 }

t = 138.457

UDESC

SUBSET-SUM

Exemplo:

S = {1, 2, 7, 14, 49, 98, 343, 686, 2.409, 2.793,
16.808, 17.206, 17.705, 117.993 }

t = 138.457

S’ = { 1, 2 , 7, 98, 343, 686, 2.409, 17.206, 117.705 }

SUBSET-SUM $\in$ NP-Completo

Passo 1: Subset-Sum $\in$ *NP*

Dado um conjunto de números inteiros S, o valor t objetivo e a solução (**certificado**) *S´*, verificar se *S´* é uma solução do problema em tempo polinomial.

```
soma = 0
Para cada s' ∈ S':
      Se s' ∉ S: retorne falso
      soma = soma + s'
Se soma != t: retorne falso
Senão: retorne verdadeiro
```

Complexidade?

SUBSET-SUM $\in$ NP-Completo

Passo 2: 3-CNF-SAT $\leq_p$ SUBSET-SUM

Dada uma fórmula ϕ instância de 3-CNF-SAT, devemos:

- Criar dois números para cada variável x_i em ϕ: v_i e v'_i
- Criar dois números para cada cláusula C_j em ϕ: s_j e s'_j

Cada número criado terá **n + k** dígitos, onde **n** é o número de variáveis e **k** é o número de cláusulas.

O valor **t** terá um valor 1 para cada dígito identificado por variável e 4 em cada dígito identificado por uma cláusula

SUBSET-SUM $\in$ NP-Completo

Passo 2: 3-CNF-SAT $\leq_p$ SUBSET-SUM

- Para cada variável v_i *e* v'_i colocamos o valor 1 no dígito identificado por x_i e 0 nos outros dígitos;
- Se o literal x_i aparece na cláusula C_j, então o dígito identificado por C_j em v_i contém valor 1;
- Se o literal $\sim x_i$ aparece na cláusula C_j, então o dígito identificado por C_j em v_i contém valor 0;
- Para cada s_j e s'_j colocamos valor 0 em todos os dígitos, com duas exceções:

  em s_j colocamos 1 no dígito C_j

  em s'_j colocamos 2 no dígito C_j

SUBSET-SUM ∈ NP-Completo

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

S = { 1, 2, 10, 20, 100, 111,
1.000, 1.011, 10.001,10.010 }

t = 11144

S' = { 10001, 1011, 111, 20, 1 }
{ $v_1, v_2, v'_3, s'_1, s_2$ }

X_1 = V, X_2 = V, X_3 = F

	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

UDESC

SUBSET-SUM $\in$ NP-Completo

Note que a maior soma de cada coluna (dígito) é no máximo 6.

Assim, para esta conversão devemos usar uma base ≥ 7.

A redução de 3-CNF-SAT para SUBSET-SUM acontece em tempo polinomial.

UDESC

Resolvendo SUBSET-SUM

É possível resolver o problema do SUBSET-SUM usando programação dinâmica com complexidade de tempo e espaço $O(nt)$, onde $\boldsymbol{n}$ é o número de elementos no conjunto e $\boldsymbol{t}$ o valor do somatório que se deseja alcançar!!

Isso significa então que **P = NP**?
(e só agora você me fala isso?!)

Resolvendo SUBSET-SUM

É possível resolver o problema do SUBSET-SUM usando programação dinâmica com complexidade de tempo e espaço $O(nt)$, onde $\boldsymbol{n}$ é o número de elementos no conjunto e $\boldsymbol{t}$ o valor do somatório que se deseja alcançar!!

Isso significa então que **P = NP**?
(e só agora você me fala isso?!)
R: Não. Existe um detalhe importante que não está sendo considerado

UDESC

Programação Dinâmica (*Subset-Sum*)

Dado um conjunto de inteiros positivos, representados como um arranjo $S[1..n]$, e um inteiro t, existe algum subconjunto de S tal que a soma de seus elementos seja t.

$$SubsetS(i,t) = \begin{cases} Verdade & \text{se } t = 0 \\ Falsidade & \text{se } t < 0 \vee i > n \\ SubsetS(i+1,t) \vee SubsetS(i+1,t-x[i]) \end{cases}$$

Exemplo: $x = \{2, 3, 5\}$ e $t = 8$.

UDESC

Programação Dinâmica (*Subset-Sum*)

SubsetSum ($x[1..n]$, t)

S[n + 1, 0] ← Verdade

para j ← 1 até t

S[n + 1, j] ← Falso

para i ← n até 1

S[i, 0] ← Verdade

para j ← 1 até $x[i] - 1$

S[i, j] ← S[i + 1, j]

para j ← $x[i]$ até t

S[i, j] ← S[i + 1, j] v S[i + 1, $j - x[i]$]

retorne S[1,t]

Programação Dinâmica (*Subset-Sum*)

Exemplo: $x = \{1, 3, 5, 7\}$ e $t = 9$.

	0	1	2	3	4	5	6	7	8	9
1	V	V	F	V	V	V	V	V	V	V
2	V	F	F	V	F	V	F	V	V	F
3	V	F	F	F	F	V	F	V	F	F
4	V	F	F	F	F	F	F	V	F	F
5	V	F	F	F	F	F	F	F	F	F

UDESC

Algoritmos que Executam em Tempo Pseudo-Polinomial

Usando programação dinâmica podemos implementar um algoritmo **pseudo-polinomial** com complexidade $O(nt)$, onde ***n*** é o número de elementos no conjunto e ***t*** o valor do somatório que se deseja alcançar!!

Como assim **pseudo-polinomial**?

Se o valor de ***t*** é limitado por um polinômio existe uma solução eficiente. Mas, se o valor de **t** for muito grande (e.g. $t = 2^n$), a solução deixa de ser eficiente, se tornando exponencial ou pior.

(Números pequenos [64 bits] vs. BigInt [n bits])

UDESC

Algoritmos que Executam em Tempo Pseudo-Polinomial

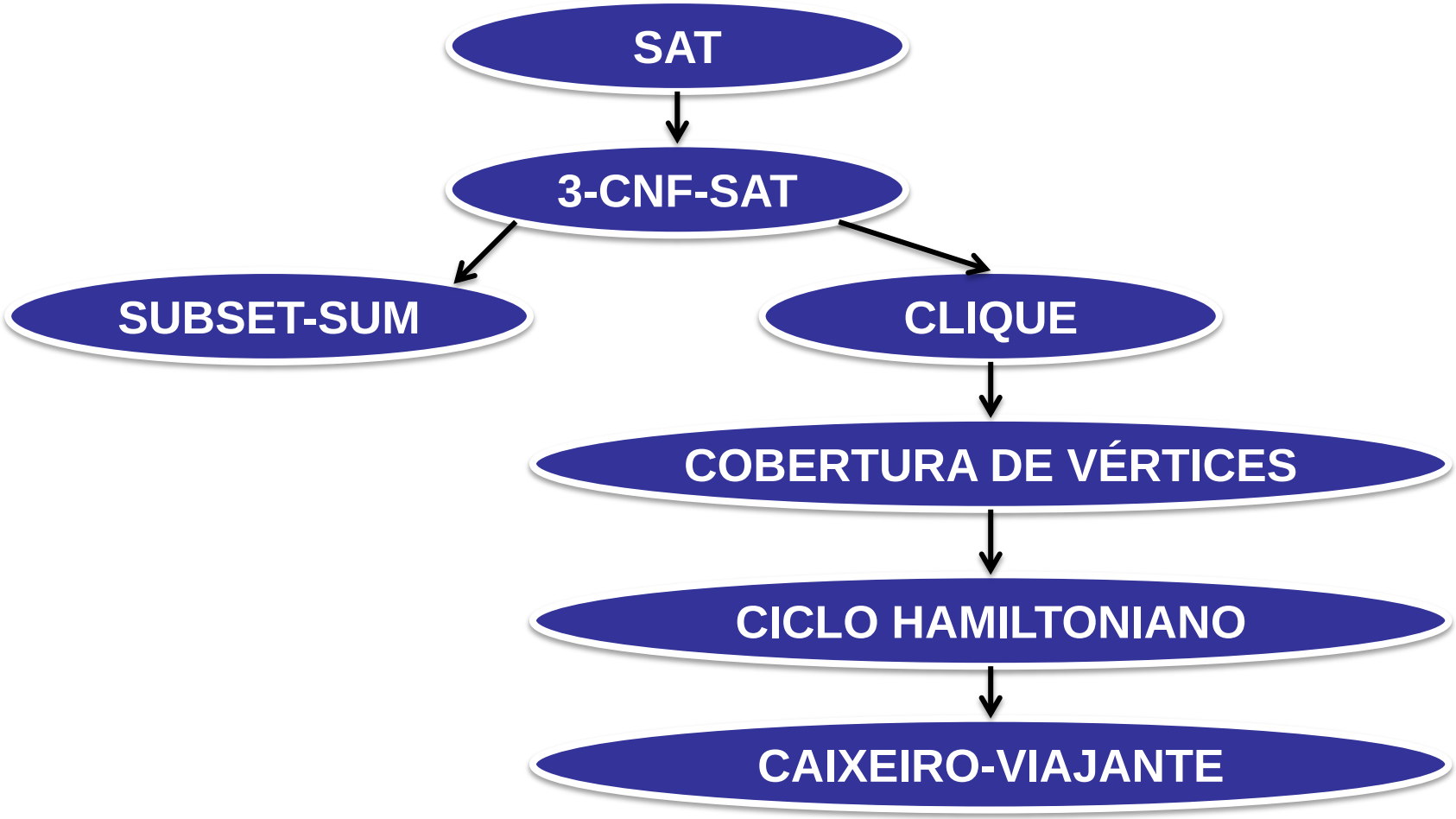
A restrição de ***t*** pequeno pode ser bastante razoável na prática:

- Problemas onde é impossível a ocorrência de números muito grandes (*e.g.* problemas de escalonamento);
- Problemas onde o tamanho do número possa ser restrito ao tamanho da palavra do processador.

Note contudo que esse não é o caso da redução do 3-CNF-SAT ao SUBSET-SUM, onde o valor de ***t*** *cresce exponencialmente em relação ao número de variáveis e cláusulas presentes na fórmula booleana.*

Reduções

Resumindo, quais reduções de problemas foram feitas:

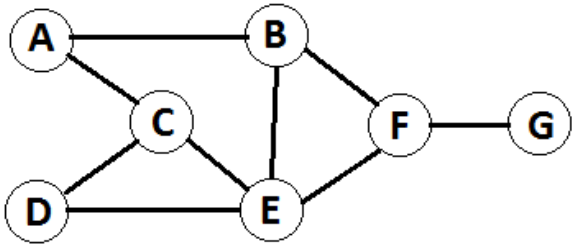


Exercícios

1) Reduzir a seguinte instância de SAT em uma instância 3-CNF-SAT:

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

2) Reduzir a seguinte instância de VERTEX-COVER em uma instância de CLIQUE. Mostre uma solução equivalente para ambos os problemas:



Exercícios

3) Reduzir a seguinte instância do problema CLIQUE para uma instância do SUBSET-SUM (CLIQUE => 3-CNF-SAT => SUBSET-SUM). Para a instância do 3-CNF-SAT, elabore uma solução válida (resultado = verdade) e outra não válida (resultado = falsidade) e as soluções correspondentes na instância do SUBSET-SUM.