* Classe NP-Completa * (Cormen, §34.3)

1. DEFINIÇÃO: um algoritmo "A" REDUZ uma linguagem L1 a uma linguagem L2 sse, para todo x1 ∈ {0,1}*, A(x1) está definido -- isto é, a execução de "A" para a entrada "x1" termina -- e x1 ∈ L1 sse A(x1) ∈ L2.

   Se "A" reduz L1 a L2, nós dizemos que L1 SE REDUZ a L2.

   Um algoritmo "A" reduz uma linguagem L1 a uma linguagem L2 EM TEMPO POLINOMIAL sse "A" reduz L1 a L2 e "A" executa em tempo polinomial (sobre o número de bits da entrada que recebe).

   Se "A" reduz L1 a L2 em tempo polinomial, então L1 se reduz a L2 EM TEMPO POLINOMIAL. Nesse caso, nós escrevemos L1 <=_P L2.

2. DEFINIÇÃO: Uma linguagem "L" é NP-DIFÍCIL sse, para toda linguagem L1 ∈ NP, L1 <=_P L.

   Uma linguagem "L" é NP-COMPLETA sse "L" é NP-difícil e L ∈ NP.

   A classe NP-completa é denotada por "NPC".

3. LEMA: se L1 <=_P L2 e L2 <=_P L3, então L1 <=_P L3.

==

PROVA:

Como L1 <=_P L2, seja A12 um algoritmo que reduz L1 a L2 em tempo polinomial.
Analogamente, seja A23 um algoritmo que reduz L2 a L3 em tempo polinomial.

Considere então o seguinte algoritmo:

```
=========================
Algoritmo: A13
Entrada: x1 ∈ {0,1}*
Saída: x3 ∈ {0,1}*
-------------------------
1. x2 := A12(x1)
2. x3 := A23(x2)
3. RETORNE x3.
=========================
```

Observe que A13 executa em tempo polinomial sobre |x1|, pois a linha 1 acima executa em tempo polinomial sobre |x1| (pois A12 é polinomial), e a linha 2 executa em tempo polinomial sobre |x2|, que, por sua vez, é polinomial sobre |x1|.

Além disso, A13 reduz L1 a L3, pois, dada uma sequência x1 ∈ {0,1}*,
x1 ∈ L1 <-> x2 ∈ L2 <-> x3 ∈ L3 (pois A12 e A23 são algoritmos de redução).

Logo, L1 <=_P L3, CQD.

==

4. LEMA: se "L1" e "L" são linguagens tais que L1 <=_P L e L ∈ P, então L1 ∈ P.

==

PROVA:

Como L ∈ P, seja "D" um algoritmo de decisão polinomial para "L". Além disso, como L1 <=_P L, seja "R" um algoritmo de redução polinomial de L1 a L. Seja, ainda, D1 o algoritmo obtido pela composição de "D" e "R", ou seja, o algoritmo

```
=========================
Algoritmo: D1
Entrada: x1 ∈ {0,1}*
Saída: 0 ou 1.
-------------------------
1. x := R(x1)
2. RETORNE D(x).
=========================
```

Observe que D1 decide L1, pois, para qualquer x1 ∈ {0,1}*, D1(x1) ∈ {0,1} e

```
  x1 ∈ L1 <-> x ∈ L         --> "R" reduz "L1" a "L"
          <-> D(x) = 1      --> "D" decide "L"
          <-> D1(x1) = 1    --> D1(x1) = D(x) .
```

Além disso, D1 sempre executa em tempo polinomial, pois, para qualquer x1, a linha 1 de D1 executa em tempo polinomial sobre |x1| (já que "R" é polinomial) e a linha 2 também (já que "D" é um algoritmo polinomial e |x| é polinomial sobre |x1|).

Logo, L1 ∈ P, CQD.

---

5. OBSERVAÇÃO: como discutido em sala, nós sabemos que P ⊆ NP, mas não sabemos se NP ⊆ P (embora a existência de numerosos e variados problemas NP-completos leve a crer que NP != P). Os resultados abaixo mostram maneiras pelas quais se poderia decidir essa questão e resultados associados.

6. COROLÁRIO: se "L" é uma linguagem NP-difícil e L ∈ P, então NP ⊆ P.

PROVA:

De fato, seja L1 ∈ NP. Como "L" é NP-difícil, então L1 <=_P L. Além disso, como L ∈ P, então, pelo lema 4 acima, L1 ∈ P. Logo, NP ⊆ P, CQD.

---

7. COROLÁRIO: se uma linguagem L ∈ NP \ P, então nenhuma linguagem NP-difícil está em P.

PROVA:

Suponha, por absurdo, que existem linguagens "L" e "L1" tais que L ∈ NP \ P, L1 é NP-difícil e L1 ∈ P. Logo, pelo corolário 6 acima (aplicado a "L1"), NP ⊆ P, e, como L ∈ NP, então L ∈ P, um absurdo.

Logo, se uma linguagem L ∈ NP \ P, então nenhuma linguagem NP-difícil está em P, CQD.

---

8. EXERCÍCIO: Mostre que, para quaisquer linguagens "L1" e "L", se "L1" é NP-difícil e L1 <=_P L, então L é NP-difícil.