* A Classe NP * (Cormen, §34.2)

0. OBSERVAÇÃO: nesta parte da disciplina, a letra "n" é frequentemente utilizada para denotar o tamanho de uma sequência binária. Em alguns pontos desta nota de aula, porém, a letra em questão é utilizada para denotar o número de vértices de um grafo.

1. DEFINIÇÃO: um algoritmo "A" VERIFICA UMA LINGUAGEM "L" sse "A" é um algoritmo de duas entradas (ambas sendo sequências binárias) e

   L = {x ∈ {0,1}* : existe y ∈ {0,1}* tal que A(x,y) = 1}.

   Nós dizemos também que "A" VERIFICA UMA SEQUÊNCIA "x" sse existe "y" tal que A(x,y) = 1. Um CERTIFICADO para uma sequência "x" (com relação a "A") é uma sequência "y" tal que A(x,y) = 1.

   Assim sendo, "A" verifica "L" sse L é o conjunto das sequências binárias que são verificadas por "A".

2. EXEMPLO: Seja VCH um algoritmo que recebe como entrada duas sequências binárias "x" e "y" e retorna "1" se "y" é (uma codificação binária de) um ciclo hamiltoniano do grafo (codificado por) "x", e que retorna "0" em caso contrário (incluindo os casos em que "x" não é uma representação válida de um grafo não-direcionado, ou em que "y" não é uma representação válida de uma sequência de vértices). Assim, se

   LCH = {x ∈ {0,1}* : "x" codifica um grafo hamiltoniano} ,

   então VCH verifica LCH, pois toda sequência "x" que representa um grafo hamiltoniano é verificada por VCH (isto é, como o grafo é hamiltoniano, então nele existe um ciclo hamiltoniano, e portanto existe um certificado "y" para "x" com relação a VCH), e toda sequência "x" que é verificada por VCH representa um grafo hamiltoniano, e portanto pertence a LCH.

3. EXERCÍCIO: considere a seguinte proposta de algoritmo de verificação para a linguagem LCH definida no exemplo anterior: um algoritmo "A" tal que A(x,y) = 1 sse "y" representa um número natural >= 1 (logo, "y" é interpretado não mais como uma sequência de vértices, mas simplesmente como um número). A ideia é interpretar "y" como o número de ciclos hamiltonianos do grafo (representado por) "x", de forma que "x" seja hamiltoniano sse o número representado por "y" for >= 1. Responda, então: "A" verifica "LCH"?

4. DEFINIÇÃO: uma linguagem "L" pertence à classe de complexidade NP sse existem um algoritmo polinomial "A" e uma constante "c" tais que

   L = {x ∈ {0,1}* : existe y ∈ {0,1}* tal que A(x,y) = 1 e |y| = O(|x|^c) } ,

   caso em que também dizemos que "A" verifica "L" EM TEMPO POLINOMIAL.

   Assim, em outras palavras, NP é a classe das linguagens que são verificadas em tempo polinomial.

   OBSERVAÇÕES:

   a) Observe que é exigido que "A" seja um algoritmo polinomial; logo, "A" sempre termina de executar, e além disso o faz em tempo polinomial sobre o tamanho da entrada recebida.

   b) Dado que "A" é exigido ser polinomial, pode-se perguntar a razão de também se exigir que |y| = O(|x|^c), isto é, que "y" tenha tamanho no máximo polinomial em relação ao tamanho de "x". A razão é que o fato de "A" ser um algoritmo polinomial garante apenas que o seu tempo de execução é polinomial SOBRE O TAMANHO DA SUA ENTRADA COMPLETA, que consiste tanto em "x" quanto em "y". Assim, como o objetivo é que "A"

verifique "x" em tempo polinomial sobre o tamanho de "x", é necessário que "y" tenha tamanho no máximo polinomial sobre "x". A título de exemplo nesse sentido, é apresentado mais adiante um problema de decisão (que chamamos de "Passeio com Número de Visitas") cujo algoritmo de verificação imediato exige certificados excessivamente grandes, não podendo ser utilizado para argumentar que (a linguagem correspondente a) o problema pertence à classe NP.

5. OBSERVAÇÃO: a seguir, exibiremos exemplos de algoritmos de verificação, os quais pressupõem algum esquema de codificação da entrada. Assim, por exemplo, no caso do problema Ciclo Hamiltoniano, nós pressupomos que o algoritmo de verificação recebe um grafo não-direcionado e uma sequência de vértices representados por sequências binárias "x" e "y", respectivamente.

   Como vimos em sala, não há apenas uma maneira de codificar tais objetos, mas sim várias delas. Para os nossos propósitos, porém, o importante é que o esquema de codificação seja tal que seja possível, a partir das sequências binárias, obter uma representação padrão dos objetos em questão no nosso modelo de computação em tempo polinomial sobre o tamanho das sequências binárias em questão, e que o inverso seja igualmente verdade, isto é, que seja possível, a partir de uma representação padrão de tais objetos no nosso modelo de computação, obter as sequências binárias correspondentes em tempo polinomial sobre o tamanho de tais objetos no nosso modelo de computação. Essa condição garante que podemos converter de uma representação para a outra em tempo polinomial. Observe que ambas as conversões são importantes:

   * Os algoritmos de decisão/verificação precisam decodificar as sequências binárias recebidas como entrada em tempo polinomial.
   * Uma codificação binária de algum objeto que não seja obtenível em tempo polinomial a partir da representação de tal objeto no nosso modelo de computação não parece ser útil na prática.

   A título de exemplo, a restrição acima exige que, dada uma sequência binária "x" que represente um grafo G = (V,E), seja possível computar uma representação de G por listas de adjacências, ou então por matriz de adjacências, em tempo polinomial sobre o número de bits de "x" (caso "x" seja uma sequência "inválida", isto é, não represente um grafo no esquema de codificação em questão, nós exigimos que esse fato seja também detectável em tempo polinomial sobre |x|). Da mesma forma, a restrição exige que, a partir de uma lista (ou matriz) de adjacências representando um grafo G = (V,E), seja possível computar a sequência binária correspondente a "G" em tempo O(n+m), onde n = |V| e m = |E|.

6. EXEMPLOS:
   a) Consideremos mais uma vez o problema Ciclo Hamiltoniano, cuja linguagem correspondente nós chamamos acima de LCH. Considere também o seguinte algoritmo:

```
==========================================================================
Algoritmo: VCH
Entrada: x,y ∈ {0,1}*
Saída: 0 ou 1.
--------------------------------------------------------------------------
1. Compute o grafo G = (V,E) representado por "x",
   ou retorne 0 se "x" não representa um grafo.
2. Compute a sequência de vértices S[1..k] representada por "y",
   ou retorne 0 se "y" não representa uma sequência de vértices.
3. Retorne 0 se k != n+1.
4. Verifique se cada vértice de "G" ocorre exatamente uma vez em S,
   exceto o vértice v = S[0], que deve ocorrer também no final da
   sequência; se isso não acontecer, retorne 0.
5. Verifique se, para todo "i", (S[i],S[i+1]) ∈ E;
   se isso não acontecer, retorne 0;
   se acontecer, retorne 1.
==========================================================================
```

Nós argumentaremos que VCH verifica LCH em tempo polinomial. De fato:

1) Para toda sequência "x" que codifica um grafo hamiltoniano, existe uma sequência "y" que representa um ciclo hamiltoniano do grafo em questão e que tem tamanho polinomial sobre |x|, e, para tal entrada (x,y), o algoritmo VCH retorna 1.

   Observação: Com relação à afirmação acima de que "y" tem tamanho polinomial em relação a |x|, observe que, embora nós não tenhamos determinado o esquema de codificação da entrada, o tamanho de uma sequência de vértices (no nosso modelo de computação) que represente um ciclo hamiltoniano é sempre menor ou igual ao tamanho do grafo em si. Logo, pela "OBSERVAÇÃO 5" acima, o tamanho de "y" é polinomial sobre o tamanho do grafo, isto é, sobre n+m, onde n = |V| e m = |E|. Finalmente, como |x| >= n+m (pois cada vértice e cada aresta precisa ser representado por pelo menos 1 bit), então |y| é polinomial sobre |x|, como desejado.

2) Para toda sequência "x" que não codifica um grafo hamiltoniano, o algoritmo VCH sempre retorna 0 (logo, em particular, não existe sequência "y" tal que A(x,y) = 1).

3) O algoritmo VCH executa em tempo polinomial sobre |x|+|y|. De fato, pela OBSERVAÇÃO 5, as linhas 1 e 2 executam em tempo polinomial sobre |x| + |y|. Além disso, a linha 3 executa em tempo constante, a linha 4 em tempo O(n) e a linha 5 em tempo $O(n^2)$. Logo, as linhas 3-5 executam em tempo $O(n^2)$, e, como |x| >= n, então as linhas 3-5 também executam em tempo polinomial sobre |x|+|y|, e portanto todo o algoritmo executa em tal tempo, como desejado.

b) Considere agora o problema de decisão Conjunto Independente, que consiste em, dados um grafo G = (V,E) e um natural "k", responder se existe em G um conjunto independente com "k" ou mais vértices. Segue abaixo um algoritmo de verificação para o problema, que utiliza como CERTIFICADO uma listagem dos vértices que formam o conjunto independente.

   (Observação: na prática, uma "listagem" de vértices é também uma "sequência" de vértices, mas nós utilizamos a primeira expressão na tentativa de enfatizar o fato de que a ordem não é importante. A palavra "lista" não foi utilizada, na tentativa de enfatizar o fato de que o candidato a certificado não precisa ser, depois de decodificado, armazenado numa lista encadeada.)

```
==========================================================================
Algoritmo: VCI
Entrada: x,y ∈ {0,1}*
Saída: 0 ou 1.
--------------------------------------------------------------------------
1. Compute o grafo G = (V,E) e o natural "k" representados por "x",
   ou retorne 0 se "x" não representa uma entrada válida para o problema.
2. Compute a listagem de vértices L[1..t] representada por "y",
   ou retorne 0 se "y" não representa uma listagem de vértices.
3. Se t < k, retorne 0.
4. Verifique se cada vértice de "G" ocorre no máximo uma vez em L;
   se isso não acontecer, retorne 0.
5. Verifique se, para todos "i" e "j" de 1 a "t", (L[i],L[j]) ∉ E;
   se isso não acontecer, retorne 0;
   se acontecer, retorne 1.
==========================================================================
```

É fácil ver que, para toda sequência x ∈ {0,1}*:

1) Se "x" representa um grafo G = (V,E) e um natural "k" tais que "G" possui um conjunto independente de "k" ou mais vértices, então existe um certificado y ∈ {0,1}* (relativo ao algoritmo VCI) para |x| de tamanho polinomial sobre |x|.

   Observação: na verdade, nós sabemos inclusive que, para o certificado "y" em questão, temos |y| <= |x|, pois a codificação de um subconjunto de vértices do grafo obviamente pode ser feita de forma a usar no máximo a quantidade de bits usada para codificar o grafo inteiro. Entretanto, esse fato não é necessário para mostrarmos que VCI verifica a linguagem do problema em tempo polinomial.

2) Se "x" não representa "G" e "k" como descrito no item anterior (seja porque "x" não representa entrada válida ou porque o grafo não possui conjunto independente de tamanho >= k), então não existe "y" tal que VCI(x,y) = 1.

Além disso, as linhas 1 e 2 do algoritmo VCI executam em tempo polinomial sobre |x| + |y| (pela OBSERVAÇÃO 5 acima), e as linhas 3 a 5 executam em tempo $O(n^2) = O(|x|^2)$, onde n = |V|. Logo, VCI verifica a linguagem do problema concreto Conjunto Independente em tempo polinomial, e portanto o problema está em NP, CQD.

7. EXERCÍCIO: seja "Passeio com Número de Visitas" o problema de, dados um grafo não-direcionado G = (V,E) e uma função f : V -> N (sendo "N" o conjunto dos números naturais), responder se existe um passeio de "G" que passa por cada vértice "v" exatamente f(v) vezes. Considere também o seguinte algoritmo:

```
================================================================================
Algoritmo: VPNV
Entrada: x,y ∈ {0,1}*
Saída: 0 ou 1.
--------------------------------------------------------------------------------
1. Compute o grafo G = (V,E) e a função f[1..n] representados por "x",
   onde n = |V|, ou retorne 0 se "x" não representa uma entrada válida.
2. Compute a sequência de vértices P[1..t] representada por "y",
   ou retorne 0 se "y" não representa uma sequência de vértices de "G".
3. Verifique se cada vértice "v" de "G" ocorre exatamente f[v] vezes em S;
   se isso não acontecer, retorne 0.
4. Verifique se, para todo "i" de 1 a t-1, (P[i],P[i+1]) ∈ E;
   se isso não acontecer, retorne 0;
   se acontecer, retorne 1.
================================================================================
```

Assim sendo:

a) Argumente que VPNV verifica a linguagem do problema em questão -- observe que não foi pedido para se mostrar que o algoritmo verifica a linguagem em tempo polinomial!

b) Argumente que o algoritmo VPNV pode ser facilmente implementado de forma a executar em tempo polinomial.

c) Argumente que, apesar dos dois fatos acima, VPNV não serve para mostrar que o problema em questão está na classe NP.

8. EXERCÍCIO: nos algoritmos de verificação acima, nós utilizamos linhas de código de alto nível -- "verifique se ..." --, apenas porque, nesta parte da disciplina, o foco está nos novos conceitos e técnicas. De fato, nós não alteramos o nosso modelo de computação, tendo apenas passado, por razões técnicas da teoria em questão, a considerar algoritmos

cuja entrada é composta de sequências binárias. Assim, os algoritmos de verificação acima poderiam ter sido escritos utilizando o mesmo nível de detalhe das partes anteriores da disciplina.

ESCREVA ENTÃO as linhas 3, 4 e 5 dos algoritmos VCH, VCI e VPNV acima no mesmo nível de detalhe que nós utilizamos anteriormente na disciplina (para recapitular o modelo de escrita anterior, consulte, por exemplo, o nosso pseudocódigo para o algoritmo de caminhos mínimos de Dijkstra).

9. EXERCÍCIO: complementando o exercício anterior, observe que as linhas 1 e 2 dos algoritmos de verificação apresentados acima também poderiam ser escritas em todo o detalhe desejado. Isso, entretanto, implicaria entrar nos detalhes particulares de algum esquema de codificação de números, grafos, etc, o que para nós é menos importante do que o conceito de verificação em si.

   ENTRETANTO, para se convencer de que a decodificação de sequências binárias é completamente factível:

   a) Escreva um algoritmo que receba como entrada uma sequência binária x[1..t] e então retorne o número natural por ela representado. Como discutido em sala, o número representado por "x" depende da codificação que se tenha em mente, mas, como "x" representa apenas um número, suponha que "x" está codificado da forma mais natural possível: em base 0, cada bit representando um dígito de "x".

      ATENÇÃO: a nossa teoria engloba sequências binárias vazias. Assim, o seu algoritmo deve tratar o caso em que "t = 0", retornando algum valor inválido nesse caso.

   b) No item anterior, você deve ter assumido que x[0] é o dígito mais significativo do número representado por "x", ou então que x[0] é o dígito menos significativo. Escreva agora um algoritmo que faz a escolha oposta.

   c) Escreva um algoritmo que recebe uma sequência binária x[1..t] que codifica DOIS números naturais "a" e "b", e que retorna "a" e "b" como saída. Você pode utilizar a codificação que lhe parecer mais conveniente, mas se assegure de utilizar uma codificação que permita que todo par de números esteja representado por alguma sequência binária.

      Por fim, existem sequências binárias que, na codificação escolhida por você, não representam nenhum par de números? Se sim, dê exemplos.

   d) Argumente que é possível representar um grafo G = (V,E) em O((n+m)*lg n) bits, onde n = |V| e m = |E|. (Dica: tanto um esquema que represente inteiros sempre em piso(lg n) + 1 bits quanto um esquema que represente cada inteiro "i" por uma quantidade de bits proporcional a "lg i" servem.)

10. TEOREMA: P ⊆ NP.

==========
PROVA:

Seja L ∈ P; nós mostraremos que L ∈ NP. De fato, como L ∈ P, existe um algoritmo polinomial "A" tal que, para todo x ∈ {0,1}*, x ∈ L <-> A(x) = 1. Para um tal algoritmo "A", seja "V" o algoritmo a seguir:

```
================================================
Algoritmo: V
Entrada: x,y ∈ {0,1}*
Saída: 0 ou 1.
------------------------------------------------
1. RETORNE A(x). // "V" simplesmente ignora "y"
================================================
```

Nós mostraremos agora que "V" verifica "L" em tempo polinomial. De fato, seja x ∈ {0,1}*; temos então:

1) Se x ∈ L, então A(x) = 1 e portanto V(x,y) = 1 para qualquer sequência binária "y", incluindo as sequências "0" e "1" (de tamanho 1) e até mesmo a sequência "" (de tamanho 0).

2) Se x ∉ L, então A(x) = 0 e portanto V(x,y) = 0 para qualquer sequência "y".

Além disso, "V" sempre executa em tempo polinomial sobre o tamanho |x|+|y| da entrada recebida, pois "V" apenas chama o algoritmo "A", que é polinomial sobre o tamanho |x| da entrada que recebe. Assim sendo, "V" verifica "L" em tempo polinomial, e portanto L ∈ NP, CQD.
==========

11. REFERÊNCIAS ADICIONAIS: como comentado em sala, a classe NP também pode ser definida em termos de algoritmos NÃO-DETERMINÍSTICOS. Uma explicação sucinta e interessante dessa definição pode ser encontrada no capítulo "http://ww3.algorithmdesign.net/sample/ch13-np.pdf" do livro "Algorithm Design. Michael T. Goodrich, Roberto Tamassia. John Wiley & Sons. 2002. ISBN: 0-471-38365-1".

    Mais detalhes podem ser encontrados no capítulo "http://www.cs.princeton.edu/theory/index.php/Compbook/Draft#np" do livro "Computational Complexity: A Modern Approach. Sanjeev Arora, Boaz Barak. Cambridge University Press. 2009. ISBN: 978-0521424264".