* NP-completude * (Cormen, cap. 34)

1. DEFINIÇÃO: um PROBLEMA ABSTRATO "PA" é uma relação binária PA ⊆ I x S, onde "I" é um conjunto de instâncias e "S" um conjunto de soluções.

2. EXEMPLOS E OBSERVAÇÕES:

   a) Formalmente, podemos definir o problema Seleção de Atividades (SA) da seguinte maneira:

      * I: o conjunto de todas as tuplas da forma

        ( (c_1,f_1), (c_2,f_2), ..., (c_n,f_n) ),

        sendo c_i e f_i números reais tais que 0 <= c_i < f_i.

        OBSERVAÇÃO: essa não é a única maneira de definir o conjunto das instâncias para o problema em questão. Em sala, por exemplo, nós consideramos a representação que utiliza pares de tuplas

        ( (c_1, c_2, ..., c_n) , (f_1, f_2, ..., f_n) ).

      * S: o conjunto de todos os subconjuntos finitos e não vazios de "N", sendo "N" o conjunto dos números naturais.

      * SA: a relação que associa cada entrada E ∈ I (que informa os tempos de início e fim de um conjunto de "n" atividades) a subconjuntos C ∈ S de índices de atividades mutuamente compatíveis, subconjuntos esses que devem possuir o maior tamanho possível para a entrada em questão.

      Assim, por exemplo, E1 = ( (1,3), (3,5), (4,5) ) é uma entrada do problema, isto é, E1 ∈ I. Já a tupla E2 = ( (1,3), (3,5,6) ) não é uma entrada válida, isto é, E2 ∉ I, pois o segundo elemento de E2 é uma tupla de 3 elementos (ao passo que os elementos das tuplas de I são todos tuplas de 2 elementos).

      Além disso, S1 = {1,2}, S2 = {1,3}, S3 = {3} e S4 = {4} são todos elementos de S, ao passo que S5 = {} não é (por ser um conjunto vazio).

      Finalmente, observe que (E1,S1) ∈ SA, isto é, S1 é uma solução (ótima) para a entrada E1, pois S1 = {1,2}, as atividades 1 e 2 de E1 são compatíveis e não existe solução para E1 que seja maior que S1. Da mesma forma, (E1,S2) ∈ SA, isto é, S2 também é solução para E1. Por outro lado, (E1,S3) ∉ SA, pois, embora S3 seja o que nós costumamos chamar de "solução viável" para a entrada E1, existem "soluções viáveis" de maior tamanho (como S1 e S2). Observe ainda que (E1,S4) ∉ SA, pois S4 = {4} e a entrada E1 não possui uma atividade de índice 4, ou seja, S4 não é sequer uma "solução viável" para E1.

      O que vimos acima ilustra, portanto, os seguintes fatos:

      1) No caso de problemas de otimização, uma entrada se relaciona apenas com as suas SOLUÇÕES ÓTIMAS (ao invés de também com as suas "soluções viáveis").

      2) Uma entrada pode se relacionar com mais de uma solução.

   b) Formalmente, o problema Ciclo Hamiltoniano (CH) pode ser definido como segue:

      (OBSERVAÇÃO: as definições sobre grafos são apresentadas em uma nota de aula anterior.)

* I: o conjunto de todos os grafos não-direcionados.

* S: o conjunto { "sim", "não" }.

* CH: a relação que associa cada grafo G ∈ I que possui um ciclo hamiltoniano à solução "sim", e que associa os demais grafos à solução "não".

3. DEFINIÇÃO: um PROBLEMA DE DECISÃO ABSTRATO é um problema abstrato PDA ⊆ I x S tal que S = {0,1} -- a solução "1" é interpretada como "sim" ou "verdadeiro", e a solução "0" como "não" ou "falso" -- e tal que PDA é uma função, ou seja,

   (E,S1) , (E,S2) ∈ PDA => S1 = S2, para todos E, S1 e S2

   (ou seja, o problema associa apenas uma solução a cada entrada).

4. EXEMPLOS: o problema Ciclo Hamiltoniano é um problema de decisão, ao passo que o problema Seleção de Atividades não o é.

5. DEFINIÇÃO: um PROBLEMA CONCRETO é um problema abstrato PC ⊆ I x S tal que I = {0,1}*, onde {0,1}* é o conjunto das sequências finitas de 0's e 1's, incluindo sequência vazia.

   Um PROBLEMA DE DECISÃO CONCRETO é uma função PDC: {0,1}* -> {0,1}.

   OBSERVAÇÃO: pela última definição, um problema de decisão concreto atribui 0 ou 1 a toda sequência de 0's e 1's. Entretanto, não necessariamente toda tal sequência representa corretamente uma entrada para o problema. A tais instâncias inválidas, nós estipulamos que o problema atribui sempre a solução "0".

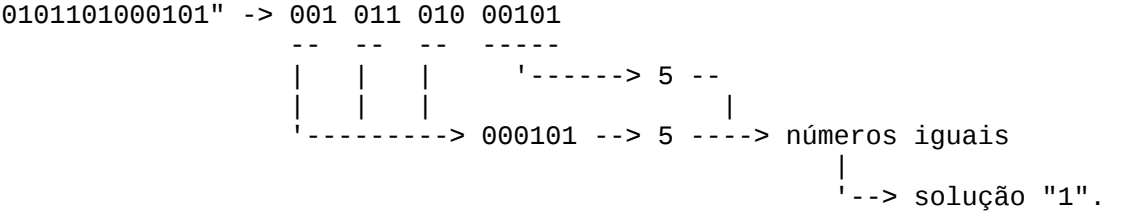
6. EXEMPLOS:

   a) Considere o seguinte problema de decisão: dados dois números naturais, responder se eles são iguais ou não. Como podemos definir uma versão concreta desse problema? Uma questão crucial é como interpretar a entrada do problema, já que, embora seja fácil interpretar uma sequência de 0's e 1's como um número natural, nesse problema uma sequência de 0's e 1's tem que representar DOIS números.

      A seguinte proposta foi feita em sala por um aluno: o primeiro número será interpretado em blocos de 3 bits -- o número 3 é arbitrário; qualquer quantidade >= 2 é suficiente --, o último dos quais informa se o número terminou -- "0" -- ou se mais um bloco de 3 bits faz parte do 1o número -- "1" --. Os bits do 2o número são todos aqueles que vierem depois do último bloco de 3 bits do 1o número. Cada número deve estar representado por pelo menos um bit.

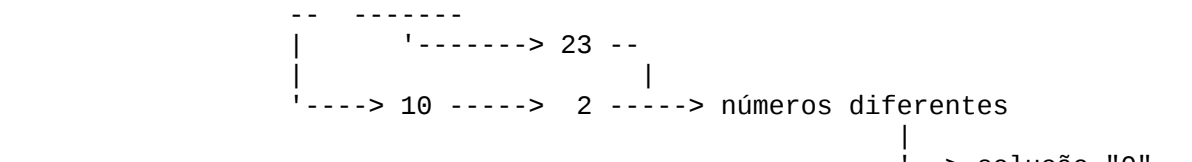
      Segue abaixo a interpretação de algumas entradas de acordo com a proposta acima, bem como as soluções correspondentes:

      * "00101101000101" -> 001 011 010 00101

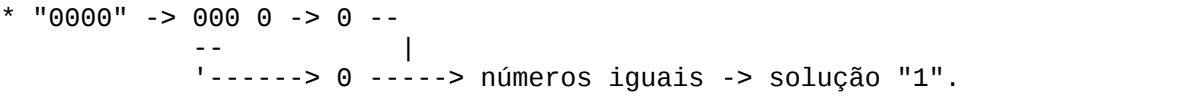



      * "" -> entrada inválida (cada número deve ser representado por pelo menos um bit) -> solução "0".

* "1000010111" -> 100 0010111




* "0000" -> 000 0 -> 0 --




* "000" -> entrada inválida (não há bits para o 2o número) -> solução "0".

Assim, o problema de decisão concreto em questão é a função que associa, a cada sequência de 0's e 1's, um elemento do conjunto {0,1}, da forma exemplificada acima.

b) O exemplo acima ilustra uma "codificação de instâncias" -- isto é, uma função de instâncias abstratas em sequências binárias -- bastante particular. Existe, porém, uma maneira geral e simples de se obter codificações. Considere, por exemplo, que um certo conjunto de instâncias abstratas pode ser facilmente representado por meio do seguinte alfabeto de 5 símbolos:

  Σ = { "0" , "1" , "{" , "}" , "," } .

Nós podemos então utilizar a função c : Σ -> {0,1}* =

  { ("0",000) , ("1",001) , ("{",010) , ("}", 011) , (",",100) }

para codificar qualquer uma das instâncias abstratas em questão, simplesmente justapondo os códigos dos símbolos de uma sequência de Σ* que represente a instância (Σ* é o conjunto de todas as sequências com símbolos de Σ).

Assim, por exemplo, consideremos a instância "v0 -- v2 -- v1" para o problema Ciclo Hamiltoniano, isto é, o grafo não-direcionado que consiste simplesmente num caminho <v0,v2,v1>. Tal instância pode ser representada assim por meio do alfabeto Σ acima:

  {{0,1,10},{{0,10},{10,1}}}

onde "{0,1,10}" representa o conjunto de vértices do grafo (lembre que "10" significa "2" na base binária) e "{{0,10},{10,1}}" representa o conjunto de arestas. (Naturalmente, "{10,1,0}" também representa o mesmo conjunto de vértices, e "{{1,10},{0,10}}" o mesmo conjunto de arestas, ou seja, a mesma instância abstrata admite diferentes representações por meio do alfabeto Σ em questão.) Assim, utilizando a função de codificação "c" acima

```
------------------------------------------------------------------------
  {   {   0   ,   1   ,   1   0   }   ,   {   {   0   ,   1   0   }   ,

  010 010 000 100 001 100 001 000 011 100 010 010 000 100 001 000 011 100

  {   1   0   ,   1   }   }   }

  010 001 000 100 001 011 011 011
------------------------------------------------------------------------
```

vemos que a instância em questão pode ser representada pela sequência binária

"0100100001000011000010000111000100100001000001000011100
0100010001000001011011011".

Logo, a versão concreta do problema Ciclo Hamiltoniano que utiliza o esquema acima para a representação das instâncias atribui "0" ("não") à sequência binária acima, já que o grafo em questão não possui um ciclo hamiltoniano.

Consideremos agora a instância "010010" para o mesmo problema. Em símbolos de Σ, ela corresponde à sequência "{{", que não representa nenhum grafo. Logo, a instância "010010" é inválida, e portanto a ela o problema atribui o valor "0".

Por fim, observe que a instância "0000" não é sequer a codificação de alguma sequência de Σ*. Logo, a ela o problema também atribui o valor "0".

7. DEFINIÇÃO: um algoritmo RESOLVE UM PROBLEMA CONCRETO em tempo O(t(n)) sse, para toda instância (concreta) de tamanho "n" (isto é, de "n" bits), o algoritmo produz a solução correspondente em tempo O(t(n)).

Assim, um algoritmo resolve um problema concreto EM TEMPO POLINOMIAL sse ele resolve o problema em tempo O(n^k), para algum natural "k" FIXO -- isto é, deve ser possível determinar "k" com base apenas na definição do problema, sem referência ao tamanho "n" da instância.

Um problema concreto é RESOLVÍVEL EM TEMPO POLINOMIAL sse existe um algoritmo que o resolve em tempo polinomial.

8. DEFINIÇÃO: uma LINGUAGEM é um subconjunto qualquer de {0,1}* -- ou seja, uma linguagem é simplesmente um conjunto de sequências binárias.

Nós dizemos que um algoritmo "A" qualquer ACEITA uma sequência binária "x" qualquer sse a saída de "A" para "x" -- que podemos denotar por "A(x)" -- é "1". Analogamente, nós dizemos que "A" REJEITA "x" sse A(x) = 0.

Uma linguagem "L" é DECIDIDA por um algoritmo "A" sse "A" aceita os elementos de "L" e rejeita as sequências de {0,1}* que não estão em "L".

Observe que toda linguagem "L" corresponde a um problema de decisão concreto, a saber, o problema que atribui "1" às sequências binárias que estão em "L" e "0" às que não estão em "L". Além disso, um algoritmo que decida "L" obviamente também resolve o problema concreto em questão. Assim, nós dizemos que uma linguagem "L" é DECIDÍVEL EM TEMPO POLINOMIAL sse o problema concreto correspondente a "L" é resolvível em tempo polinomial.

9. DEFINIÇÃO: a CLASSE DE COMPLEXIDADE "P" é o conjunto dos problemas DE DECISÃO CONCRETOS que são resolvíveis em tempo polinomial, ou, de forma equivalente, o conjunto das linguagens decidíveis em tempo polinomial.

10. EXERCÍCIOS: utilizando o modelo de computação com que utilizamos durante toda a disciplina, responda se os seguintes problemas pertencem à classe de complexidade P:

a) O problema de responder se um dado grafo possui um caminho de no máximo "k" arestas entre dois dados vértices "u" e "v".

b) O problema de responder se uma dada entrada para o problema da mochila 0-1 possui solução ótima de valor >= a um certo valor "k".

11. DEFINIÇÃO: uma linguagem "L" é ACEITA por um algoritmo "A" sse, para toda sequência x ∈ {0,1}*,

A(x) = 1 <-> x ∈ L.

Em outras palavras, "L" é aceita por "A" sse "L" é o conjunto das sequências para as quais o algoritmo retorna "1".

**IMPORTANTE**: se "L" é aceita por "A", então, para cada x ∉ L, há duas possibilidades: ou A(x) = 0 -- isto é, o algoritmo termina e retorna "0" -- ou A(x) não é definido -- isto é, o algoritmo não termina.

Por fim, nós dizemos que "L" é aceita EM TEMPO POLINOMIAL por "A" sse "L" é aceita por "A" e existe um polinômio t(n) tal que, para toda instância x ∈ L de "n" bits, "A" retorna a solução para "x" (que nós sabemos ser A(x) = 1) em tempo O(t(n)).

**IMPORTANTE**: observe que a definição anterior não faz menção ao tempo levado por "A" para instâncias fora de "L"! Assim, suponha que um algoritmo "A" aceita uma linguagem "L" em tempo polinomial, e que, para alguma constante "c" e algum polinômio t(n), toda saída A(x) = 1 é retornada por "A" em tempo <= c*t(|x|), onde |x| é o número de bits de "x". A observação é então a de que pode existir uma sequência x ∈ {0,1}* para a qual o algoritmo termina mas leva tempo mais que c*t(n) para retornar a saída; a única ressalva é que isso somente pode acontecer se A(x) = 0, isto é, se x ∉ L.

12. TEOREMA: P é igual ao conjunto das linguagens que são aceitas em tempo polinomial.

==========

PROVA:

Por definição, temos que

P = {L ⊆ {0,1}* : L é decidida em tempo polinomial}.

Nós temos então que mostrar que P = AC, sendo

AC = {L ⊆ {0,1}* : L é aceita em tempo polinomial}.

Mostremos primeiramente que P ⊆ AC. De fato, seja L ∈ P. Logo, existe um algoritmo polinomial "A" tal que, para toda sequência binária "x",

* x ∈ L => A(x) = 1;
* x ∉ L => A(x) = 0.

Temos que mostrar que L ∈ AC. Logo, temos que mostrar que existe um algoritmo "B" tal que, para toda sequência "x", B(x) = 1 sse x ∈ L, e tal que toda saída 1 retornada por "B" é dada em tempo polinomial. Claramente, o algoritmo "A" satisfaz o requisito em questão, e portanto L ∈ AC, CQD.

Temos agora que mostrar que AC ⊆ P. Seja, então, L ∈ AC. Logo, existe um algoritmo "A" tal que, para toda sequência "x", x ∈ L sse A(x) = 1. Além disso, existem constantes "c" e "k" tais que toda saída 1 dada pelo algoritmo é retornada por ele após a execução de no máximo c*n^k instruções. Considere, então, o algoritmo "B" que recebe uma sequência "x" e então executa as mesmas instruções que "A", exceto que ele também conta as instruções executadas, e sempre pára após executar c*n^k instruções de "A", retornando, nesse caso, o valor 0. Informalmente, nós podemos dizer que "B" "simula", em parte, a execução de "A": se "A" retorna uma solução após executar c*n^k instruções ou menos, então "B" retorna a mesma solução; se, porém, "A" não retorna nenhuma solução após executar c*n^k instruções, então "B" retorna "0".

Nós argumentaremos agora que o algoritmo "B" decide "L" em tempo polinomial. De fato, observe primeiramente que, para toda sequência de entrada "x", "B" retorna um valor de saída B(x), que é ou 0 ou 1. Os casos possíveis são então:

* B(x) = 1: logo, pela definição de "B", A(x) = 1 (isto é, se "B" retorna "1", então "A" retorna 1 após no máximo c*n^k instruções). Logo, pela suposição inicial sobre "A", temos que x ∈ L.

* B(x) = 0: dois casos são possíveis:

  a) "B" retorna zero para a entrada "x" porque "A" retorna zero para "x" após executar c*n^k instruções ou menos: nesse caso, pela suposição inicial sobre "A", temos que x ∉ L.

  b) "B" retorna zero para a entrada "x" porque "A" não retorna nenhuma solução após executar c*n^k instruções: logo, não é possível que x ∈ L, pois, se esse fosse o caso, então, pela suposição inicial sobre "A", "A" retornaria 1 para a entrada "x" após executar no máximo c*n^k instruções, o que não acontece. Logo, x ∉ L.

A análise de casos acima mostra então que, para toda sequência "x", B(x) está definido, valendo 1 se x ∈ L e 0 se x ∉ L. Logo, "B" decide "L".

Para concluir o caso AC ⊆ P, resta apenas mostrar que "B" é um algoritmo polinomial, o que segue do fato de "B" sempre retornar uma solução após a execução de no máximo c*n^k instruções de "A", e do fato de a contagem de instruções realizada por "B" não alterar a complexidade do algoritmo, no sentido de que "B" executa em tempo O(n^k). Logo, "B" decide "L" em tempo polinomial, o que implica que AC ⊆ P, CQD.
==========