Linguagens Variáveis no Tempo e o Problema P x NP

(Variables Languages in Time and the Problem P x NP)

Valdir Monteiro dos Santos Godoi
valdir.msgodoi@gmail.com

Conforme sabemos, não é possível acertar 100% das vezes nossas previsões sobre o resultado de um jogo de par ou ímpar, cara ou coroa, lançamento de dados, roleta, cartas, loteria, mercado de ações, etc., enfim, todos os eventos onde o caráter probabilista domina, tem forte influência.

Não é possível acertar sempre nossas previsões no sentido determinista, admitindo-se que vivemos num mundo onde existe o livre arbítrio. Embora existam as rígidas leis da Física, elas não são capazes de predizer o futuro em toda a sua extensão, e com toda a precisão.

Sendo assim, está condenada ao fracasso qualquer tentativa de construir um algoritmo ou programa de computador determinísticos, destinados a acertar inequivocamente, sem erro algum, e em todas as vezes, estes resultados probabilísticos ou aleatórios (desde que não se trate de dados viciados, cartas marcadas, times e juízes comprados, etc.).

O mesmo já não pode ser dito de algoritmos não determinísticos. Um algoritmo não determinístico pode ser considerado como um processo que, quando confrontado com uma escolha entre duas (ou mais) alternativas, pode criar cópias de si mesmo para cada alternativa e prosseguir o processamento para cada uma delas, independentemente das demais, em paralelo. Se existir um conjunto de possibilidades que levem a uma resposta positiva então esse conjunto é sempre escolhido e o algoritmo terminará com sucesso ([1], [2], [3]).

Num exemplo simples de lançamento de dados, onde os resultados possíveis são os elementos do conjunto DADOS = {1, 2, 3, 4, 5, 6}, nossa máquina (de Turing) determinística (ou computador moderno) poderá lançar seu palpite entre os elementos de DADOS, mas terá apenas 1/6 de probabilidade de acerto.

Como nossa idealizada máquina de Turing não determinística poderá escolher cada uma das seis alternativas possíveis de DADOS (produzindo, digamos, seis cópias de si mesmo), uma das alternativas deverá evidentemente coincidir com o resultado correto do lançamento do dado, e o processamento terminará no estado de sucesso ou aceitação.

Vimos assim que uma máquina ou algoritmo não determinístico são capazes de algo que a correspondente versão determinística não é capaz: acertar sempre.

Para cada lançamento a linguagem relacionada ao resultado correto variará em função deste resultado, ou seja, se nosso dado mostrou a face 1 para cima então a linguagem L = {1} é a linguagem que produzirá o resultado correspondente ao sucesso/aceitação naquele momento, enquanto os demais valores possíveis, 2, 3, 4, 5 e 6, não pertencerão a L durante aquela jogada específica, com aquele jogador específico.

Num momento seguinte poderemos ter L = {3}, no próximo lançamento e a seguir L = {6}, depois L = {1}, L = {5}, etc., evidenciando que temos um exemplo de linguagem não constante, e variável no tempo, dependente de cada nova situação.

Para que um algoritmo não determinístico, ou sua correspondente MTND, construídos para reconhecer e aceitar estas linguagens sejam capazes de decidir entre aceitar ou rejeitar um dado de entrada, entre informar um SIM ou NÃO, SUCESSO ou INSUCESSO, é necessário que venham dados do ambiente externo, a fim de se poder decidir, pela comparação, sobre a aceitação ou não de seu palpite, previsão, estimativa, etc.

Se definirmos uma linguagem da forma

$L = \{w = (x\ y\ z); x$ é a chave do problema, y é o palpite da máquina, calculado e registrado antes de se saber o valor de z, z é o resultado correto do problema$\}$

a máquina aceitará sempre que $y = z$, e rejeitará caso contrário.

Este é um procedimento em tempo polinomial: dado x gera-se y (de maneira não determinística para as MTND ou determinística para as MTD), obtém-se z (do ambiente externo) e se $y = z$ aceita-se a entrada w.

Uma MTND, corretamente projetada, tenderá por aceitar um de seus palpites y, pois gerará todos os valores possíveis para z, um para cada string de entrada, então esta é uma classe de problemas que pertence a NP.

A versão determinística não terá a "habilidade", propriedade, de produzir cópias de si mesma, como se fossem processamentos em universos paralelos e simultâneos, e poderá apenas fornecer um único palpite para a chave x, que seja baseado em algum tipo de procedimento matemático e/ou estatístico mais adequado para a solução da questão. Dado o caráter incerto destes problemas (dados, roletas, cartas, bolsa de valores, adivinhações, etc.) não se poderá dizer que se construiu um algoritmo, nem máquina determinística, para se acertar/resolver o problema, com certeza absoluta, sendo assim estes tipos de problemas não pertencem a P, mas pertencem a NP, então $P \neq NP$.

Vemos que esta é uma demonstração que utiliza a mais fundamental propriedade do não determinismo, uma característica que as máquinas determinísticas são incapazes de realizar, que é a produção de "cópias" de si mesma e o processamento simultâneo com as outras "versões" da máquina (ou programa), até mesmo um número exponencial de vezes em relação ao tamanho da entrada de cópias de si mesma.

Se preferirmos não adotar esta propriedade de criação e paralelismo, e ao invés disso admitirmos que as MTND têm uma sorte absoluta, que são abençoadas com uma sorte inacreditável, de modo que sempre fazem a melhor escolha [4], na primeira tentativa, como parece defender Papadimitriou *et al* em [5], nossa demonstração não mudaria em essência: o algoritmo não determinístico acertaria sempre, desta vez por sorte extrema, mesmo sem criar novas versões da máquina, uma para cada alternativa que é necessária ao algoritmo. O algoritmo determinístico, por sua vez, não teria nenhuma sorte absoluta, faculdade

premonitória, nem poder "sobrenatural" de multiplicação instantânea, e só seria capaz de acertar, em geral, em termos probabilísticos.

Certamente que tem de ser $P \neq NP$.

Referências

[1] R.M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York (1972), 85–103.

[2] J.E. Hopcraft, J.D. Ullman and R. Motwani, *Introdução à Teoria dos Autômatos, Linguagens e Computação*, Rio de Janeiro: Elsevier e Campus (2003), 452.

[3] N. Ziviani, *Projeto de Algoritmos com Implementações em Java e C++*, São Paulo: Thomson Learning (2007), 381-383, 388, 551.

[4] Devlin, Keith, *Os Problemas do Milênio – sete grandes enigmas matemáticos do nosso tempo*, Rio de Janeiro: editora Record (2004).

[5] S. Dasgupta, C. Papadimitriou and U. Vazirani, *Algoritmos*, São Paulo: McGraw-Hill Interamericana do Brasil Ltda. (2009) 244.