RISI **1**(1), 102-105 (2004)
Rev. investig. sist. inform.
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos
ISSN: 1815-0268 (impreso)

E N S A Y O S

TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL Y TEORÍA DE LA COMPUTABILIDAD

Augusto Cortéz*

RESUMEN

La ciencia de la computación es un cuerpo sistematizado del conocimiento concerniente al cálculo, que se sostiene en dos áreas fundamentales: La Teoría de la Computabilidad, basada en las ideas y los modelos fundamentales subyacentes al cálculo, y las técnicas de la ingeniería para el diseño de algoritmos. Este artículo está pensando en la importancia del primer aspecto. La Teoría de la Complejidad computacional estudia los recursos requeridos para resolver un problema como son el tiempo y el espacio; por su parte la teoría de la computabilidad se interesa en expresar los problemas como algoritmos sin tener en cuenta la información sobre los recursos necesarios para ello.

Para abstraer las variaciones entre los diferentes sistemas computacionales se utiliza una máquina de Turing como un referente fijo, considerado como un modelo de máquina isofórmica a cualquier otro sistema informático.

La Tesis de Church-Turing nos dice que si la máquina de Turing no puede resolver un problema, ninguna otra computadora podrá hacerlo, puesto que no existe algoritmo para resolver el problema. Por esa razón, las limitaciones corresponderían a los procesos computacionales y no a la tecnología.

Palabras clave: Complejidad computacional, computabilidad, eficiencia de algoritmos.

THEORY OF THE COMPLEXITY CALCULUS THEORY OF THE CALCULABLE

ABSTRACT

The computation's science is a systematized body of the knowledge concerning to the calculus, which is supported in two fundamentals areas: The theory of the calculable supported in the ideas and fundamentals models underlying to the calculus, and the engineering's technical for the algorithm's sketch.

This article is thinking in the importance of the first aspect. The complexity's theory calculable study the resource required during the calculus to resolve a problem as the time and the space; for its part the theory of the calculable employs of the feasible to express problems as effective algorithms besides to take under advisement the necessaries resources for them.

To do the variations between different of the computers systems, is necessary to use Turing machine as permanent referring, considerate as a isomorphic machine's model to any other computers system.

The Church-Turing's thesis said that if a Turing's machine can't resolve a problem, any computer would do, because there isn't algorithm to resolve a solution. For that reason the limitations corresponds to process of computers and not to the technology.

Key word: Computacional complexity, computabilidad, efficiency of algorithms

* Docente de la Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos, Lima-Perú. E-mail: acortezv@unmsm.edu.pe

INTRODUCCIÓN

La ciencia de la Computación es el cuerpo sistematizado del conocimiento concerniente al cálculo. Sus principios se remontan al diseño de algoritmos por parte de Euclides , al uso de la complejidad asintótica y la reducibilidad por parte de los babilónicos. La ciencia de la Computación se sustenta en dos áreas básicas: los conceptos fundamentales subyacentes al cálculo que trajo como consecuencia la existencia de funciones probablemente no calculables o computables, y por otra parte, las técnicas de ingeniería para el diseño de sistemas de computación basadas en técnicas de diseño de algoritmos: algoritmos voraces (divide y vencerás), algoritmos aleatorizados, algoritmos con retroceso, etc. Este artículo está pensado en subrayar la importancia de la primera área, al relacionar los aspectos teóricos que se presentan en el estudio de la ciencia de la Computación y los aspectos prácticos con los que los estudiantes están familiarizados. Los estudiantes con conocimientos en programación, estructura de datos e ingeniería de *software* presentan problemas para establecer una relación con otros temas como: teoría de autómatas, lenguajes formales y funciones recursivas. Se tiene como objetivo resaltar la necesidad de enseñar a los alumnos habilidades para una buena programación y simultáneamente capacidad de análisis de algoritmos, con la finalidad de dotarlos de una infraestructura lógico matemática, sobre la cual fundamentar los conceptos e ideas de complejidad y que ésta se relacione y/o vincule estrechamente a los temas prácticos. No debe sorprendernos el hecho de que las ideas básicas de la teoría de autómatas se desarrollaron para resolver cuestiones teóricas de los fundamentos de las matemáticas, como lo estableció el matemático alemán David Hilbert (1862-1943).

Un proceso computacional, también llamado proceso algorítmico o algoritmo, es fundamental para la ciencia de la Computación, puesto que un computador no puede ejecutar un problema que no tenga una solución algorítmica. Así, cualquier limitación de las capacidades de los procesos computacionales constituye también una limitación de las capacidades de la computadora [1]. Evaluar la eficiencia de los algoritmos tiene mucho que ver con evaluar la complejidad de los mismos. Aunque esta labor pueda resultar ardua, compensa el hecho de obtener programas con alta garantía de corrección y la satisfacción intelectual de haberlos construido bien desde el principio, sin esperar el veredicto del sistema computacional. Sin embargo, el tema que nos aborda es establecer si una función es calculable o computable, soslayando las especificaciones de un sistema computacional especifico. En 1935, el matemático y lógico inglés Alan Mathison Turing (1912-1954) se intereso en el problema de decisión de Hilbert, que preguntaba si podría existir un método general aplicable a cualquier enunciado, para determinar si éste era verdadero. El enfoque de Turing lo llevó a pensar lo que ahora se conoce como la máquina de Turingn [2]. El tema de la computabilidad tiene mucho que ver con la búsqueda de las estructuras que se requieren en un lenguaje de programación, de tal forma que se asegure que un programa expresado en dicho lenguaje pueda resolver cualquier problema que tenga una solución algorítmica [4].

ANÁLISIS DE ALGORITMOS

La Teoría de la Complejidad Computacional es la parte de la teoría de la computación que estudia los recursos requeridos durante el cálculo para resolver un problema. Un cálculo resulta complejo si es difícil de realizar. En este contexto podemos definir la complejidad de cálculo como la cantidad de recursos necesarios para efectuar un cálculo. Así, un cálculo difícil requerirá más recursos que uno de menor dificultad. Los recursos comúnmente estudiados son el *tiempo* (número de pasos de ejecución de un algoritmo para resolver un problema) y el *espacio* (cantidad de memoria utilizada para resolver un problema). Un algoritmo que resuelve un problema pero que tarda mucho en hacerlo, difícilmente será de utilidad. Igualmente un algoritmo que necesite un gigabyte de memoria no será probablemente utilizado. A estos recursos se les puede añadir otros, tales como el número de procesadores necesarios para resolver el problema en paralelo. Si un cálculo requiere más tiempo que otro decimos que es más complejo y lo llamamos complejidad temporal. Por otro lado, si un cálculo requiere más espacio de almacenamiento que otro decimos que es más complejo, en este caso hablamos de una complejidad espacial. Es claro que el tiempo que se requiere para efectuar un cálculo en un computador moderno es menor que el requerido para hacer el mismo cálculo con las máquinas de las década pasadas, y seguramente que el tiempo en una máquina de la próxima generación será mucho menor [1].

El estudio de los procesos computacionales, conduce a una clasificación de los problemas en dos grandes clases: los problemas con solución y los problemas sin solución. Los problemas

solucionables, requieren gran cantidad de recursos como tiempo y espacio de almacenamiento. El análisis requerido para estimar el uso de recursos de un algoritmo es una cuestión teórica, y por tanto necesita un marco formal[5]. El formalismo y abstracción constituye un método necesario para la programación, considerada como disciplina científica en la que se prioriza la actividad de diseño y razonamiento, sobre la de depuración mediante prueba y error en un contexto artesanal.

Los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño. Aunque actualmente la mayoría de los algoritmos resueltos por las máquinas tienen como máximo una complejidad o costo computacional polinómico, es decir, la relación entre el tamaño del problema y su tiempo de ejecución es polinómica. Éstos son problemas agrupados en la clase P. Los problemas con costo no polinomial están agrupados en la clase *NP.* Estos problemas no tienen una solución algorítmica, es decir, una máquina no puede resolverlos en un tiempo razonable.

Existe una escala para medir la complejidad, la que incluye, entre otros: P, Resoluble en tiempo polinómico; P-completo, los problemas más difíciles en P. NP problemas con respuestas positivas verificables en tiempo polinómico y NP-completo, los más difíciles problemas de NP. Muchas de estas clases tienen una *co-clase* que contiene los problemas complementarios a los de la clase original. Por ejemplo, si X está en NP, el complemento de X está en co-NP. Sin embargo, no debe entenderse que NP y co-NP sean complementarios; hay problemas que pertenecen a ambas clases, y otros que no están en ninguna de los dos.

Cada cierto tiempo se duplica el número de instrucciones por segundo que los computadores pueden ejecutar. Los profanos y no tan profanos piensan que esto puede inducir a muchos programadores a pensar que basta esperar algunos años para que problemas que hoy necesitan muchas horas de cálculo puedan resolverse en pocos segundos, soslayando la necesidad de evaluar y obtener algoritmos eficientes. Este artículo pretende deshacer tal suposición, mostrando que el factor predominante que delimita lo que es soluble en un tiempo razonable de lo que no lo es, es precisamente el algoritmo elegido para resolver el problema. El hombre es creativo por naturaleza, la tecnología le ofrece un mundo de posibilidades de exploración, por tanto, no debe sustraerse de los beneficios que ésta le provee. Particularmente, el advenimiento de nueva tecnología debe tomarse como un estímulo para buscar algoritmos más eficientes.

COMPUTABILIDAD

Es conveniente suponer que un sistema computacional es fijo (abstracto), con la finalidad de eliminar las variaciones de un sistema a otro. Brookshear elimina estas variaciones de análisis en sistemas diferentes y evalúa la complejidad de cálculo suponiendo un sistema fijo basado en una máquina de Turing, ya que supone que a partir de ella puede trasladar sus resultados a otros sistemas y mantener la complejidad.

Un lenguaje de programación (el preferido por usted) tiene un poder expresivo ya que permite escribir cualquier algoritmo (una idea) en ese lenguaje. Sin embargo, suponemos que existen algunos algoritmos que no pueden representarse haciendo uso de ese lenguaje. Surge la interrogante de si esto se debe a una limitación del lenguaje o es una limitación de los procesos algorítmicos en general (no existe algoritmo). Nos preguntamos, si el lenguaje no permite escribir un algoritmo o es que no existe un algoritmo que pudiera calcular dicha función. Consideremos una clase que contiene todas las funciones computables. Una función es computable si existe un algoritmo para ella, sin importar como pueda implantarse o expresarse ese algoritmo. Puede construirse un conjunto que contenga todas las funciones computables, para ello se utiliza un método muy usado por los matemáticos, el método recursivo generacional. Este método permite que a partir de un conjunto básico de funciones computables, consideradas como funciones iniciales, puedan construirse las demás combinándolas en forma recursiva y así obtener el conjunto mayor de todas las funciones computables. Puede demostrarse, aunque no es el objetivo de este artículo, que una función de esta clase puede calcularse por medio de una máquina de Turing.

La Teoría de la Complejidad Computacional se basa en un enfoque no funcional, en donde interesa como se lleva el cálculo y su complejidad en función de los recursos que utiliza. La Teoría de la Computabilidad se interesa en hallar una solución a un problema, mas no se interesa por los métodos específicos de expresar una solución algorítmica en un sistema computacional; es decir, no interesa cómo se pueda implantar.

HIPÓTESIS DE CHURH

La hipótesis de Churh nos dice que una función computable puede identificarse con la clase de las funciones parcialmente recursivas. Una función parcialmente recursiva F: X ➔ X es aquella cuyo dominio constituye un subconjunto de X, por ejemplo en la función div: Zx Z ➔ Z, el par (x, 0) no está en el dominio de F, puesto que no se puede dividir entre cero. La base de la jerarquía de estas funciones computables consiste en un conjunto de funciones iniciales, a partir de las cuales se construyen las funciones recursivas primitivas mediante composición de funciones. Todas estas funciones son computables, sin embargo, existen funciones que no son primitivas pero son computables, de estas funciones se dice que no son totales sino parcialmente recursivas. Inicialmente se pensó que todas las funciones recursivas primitivas eran computables, pero Ackerman propuso la conocida función que lleva su nombre A(0,y)=y+1, A(x+1,0)=A(x,1) y A(x+1,y+1)=A(x, A(x+1,y)), la cual es computable pero no es recursiva primitiva. Como esta hipótesis no pone límites al número de pasos o cantidad de almacenamiento necesario para la función, supone que las funciones parcialmente recursivas son computables. Una función no es computable a menos que pueda limitarse el cálculo por adelantado o determinar que el cálculo terminará o no en algún momento [3]. Las máquinas de Turing constituyen una herramienta de verificación, pues además de servir como aceptadores de lenguajes formales, pueden considerarse como una calculadora de funciones de los enteros a los enteros. Las funciones calculadas por una máquina de Turing se conocen como funciones parcialmente recursivas, y éstas incluyen a las funciones computables.

CONCLUSIÓN

Los procesos computacionales están limitados por el lenguaje en el cual puedan expresarse. La Teoría de la Complejidad estudia la eficiencia de los algoritmos en función de los recursos que utiliza en un sistema computacional (usualmente espacio y tiempo). La Teoría de la Computabilidad pretende abstraer los detalles de los sistemas computacionales y busca un algoritmo para efectuar un cálculo sin preocuparse por los detalles de implantación; en este caso se dice que la función es computable o calculable. Puede verificarse si una función es computable utilizando una máquina de Turing como un modelo de máquina isomorfa a cualquier otro sistema computacional. El interés de Turing acerca de la capacidad de las máquinas para pensar lo llevó a desempeñar un papel importante en el desarrollo de las computadoras, no solo teóricas sino reales. Así, se refuerza la tesis de Church-Turing que dice que si una máquina de Turing no puede resolver un problema, entonces ningún computador puede hacerlo, ya que no existe algoritmo para obtener una solución. Por tanto, las limitaciones corresponden a procesos computacionales y no a la tecnología.

REFERENCIAS BIBLIOGRÁFICAS

1. [BROOKSHEAR] BROOKSHEAR J. Glean. *Teoría de la computación.* Addisson Wesley iberoamericana Wilmington Delaware 1993.
2. [GRIMALDI] GRIMALDI, Ralph. *Matemática discreta y combinatoria.* Addisson Wesley iberoamericana Wilmington Delaware 1997
3. [HOPCROFT 1993] Hopcroft Jhon, Ullman Jeffrey. *Introducción a la teoría de autómatas.* Edit. CECSA 1993.
4. [PEÑA] PEñA MARI, Ricardo. *Diseño de programas*. *Formalismo y abstraccion*. Prentice Hall Madrid 1998.
5. [WEISS] WEISS Mark Allen. *Estructura de datos y algoritmos.* Addisson Wesley iberoamericana Wilmington Delaware 1995.

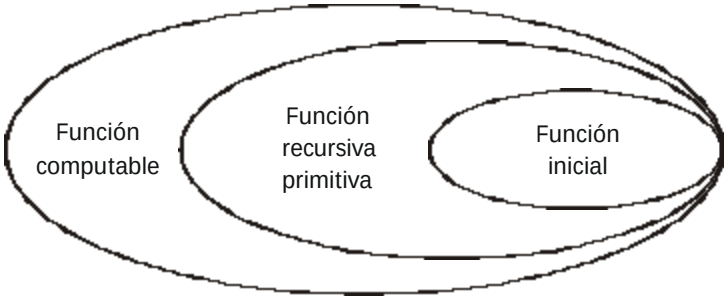

Figura N.º 1. Jerarquía de funciones computables.