Matemáticas de Cómputo = Cómputo de Matemáticas

Andreas Polymeris

Introducción

En el último número de nuestra *Revista* --en un ensayo titulado *¡Cómputo! No Matemáticas*-- interrogábamos el rol de las matemáticas en computación: ¿Son *teoría* que se limita a *hablar de computación*? ¿O *participan* directamente en la *práctica computacional*?

Para ello nos pareció interesante remitirnos a la prehistoria de estas cuestiones, visualizar cómo los recuentos de ovejas, que en base a guijarros de diferentes tamaños computaban los antiguos pastores, dieron, primero, lugar a la *notación posicional* que hoy usamos, y sólo luego, a los *números naturales* que denotamos de esa manera, es decir, a las *abstracciones* matemáticas que hoy entendemos que nuestras notaciones *representan*.

Decíamos que el único defecto que tenían estas notaciones basadas en guijarros o inscripciones en tablillas de cera, era su *peso*: entendíamos por eso, que fue la *comunicación* la que exigió *tipos de datos* no sólo más universales, sino esencialmente más *livianos*. Si un pastor quería ofrecerle su rebaño al vecino --que, como hoy en el extremo sur de nuestro país, moraba a gran distancia-- no resultaba práctico tener que enviar tanto guijarro o tablillas.

Por todo eso es que concluíamos en nuestra primera revisión de este asunto, que los números que hoy usamos, así como todas las *comprensiones matemáticas* de relevancia computacional, habrían surgido a posteriori, en la *conversación*, con el *hablar de cómputos*; pero que la *práctica computante* misma, bien puede prescindir de ellas.

¿Será tan así? nos preguntamos hoy nuevamente.

Pero esta vez enfocaremos la cuestión a partir uno de los más modernos vínculos que se han venido dando entre computación y matemáticas: la llamada *teoría de la complejidad algorítmica* que desde hace unos veinte años da trabajo a casi la mitad de los matemáticos que *hablan* de computación.

Pero no se asusten, que a pesar de que voy a *hablar* de potencialidades computacionales, aunque sea sólo para intentar una *comunicación* diferente, trataré de no usar matemáticas; es decir, intentaré *contar* el cuento al estilo computacional: *con guijarros*.

Pero antes de sumergirnos por un buen rato en las profundidades de esta *teoría de la complejidad*, quiero --para animarlos a acompañarme en este buceo-- delatarles lo que habremos *computado* cuando volvamos a la superficie. Exhibiremos un argumento contundente que nos permitirá rebatir mala parte de lo que resumí en los párrafos anteriores: en el fondo --de nuestro buceo-- demostraremos que todo cómputo usa matemáticas, aún si el *tipo de datos* que emplea no goza del reconocimiento público, universal, que por ejemplo tienen los *números naturales*. Porque entenderemos que esos *tipos de datos*, más que *representar* realidades como los rebaños de ovejas, son *comprensiones* de ellas: *conocimiento que habla* de las *cosas* a las que se refiere el cómputo.

Más no puedo revelar en seco; así que no vacilemos: zambullámonos en las *complejidades algorítmicas*.

Complejidad

La cuestión principal que esta teoría tematiza, es la siguiente: Dada alguna *tarea computacional*

genérica, ¿puede esta ser *ejecutada eficientemente*? A saber: ¿Es posible concebir un *algoritmo computacional* que para toda *instancia* de la tarea planteada --para todo *input* que *especifica lo genérico*-- efectúe lo que la tarea exige en un *tiempo de cómputo razonable*.

Por ejemplo: ¿Puede nuestro pastor *sumar dos recuentos* eficientemente? Ya sabemos que lo que él haría con sus guijarros es lo mismo que nosotros hacemos cuando sumamos dos números naturales *representados* de acuerdo a la notación posicional con base 10: Tal vez primero preprocesar la instanciación --los datos que especifican los dos números a sumar-- recorriendo las dos notaciones de derecha a izquierda, complementando la más *corta* con ceros, hasta la última posición usada por la notación más *larga*. Así logramos que las dos notaciones ahora tengan el mismo *largo* λ, número que durante este preprocesamiento aprovechamos de determinar. Luego podemos efectuar el procesamiento propiamente tal, usando para ello una *variable auxiliar* que sólo asumirá valores en {0,1}--que nuestro pastor puede aterrizar en un guijarro *tipo especial*-- e inicialmente tiene el valor 0. Para ello volveremos a recorrer las dos notaciones, de derecha a izquierda, sumando en cada iteración los dos correspondientes dígitos de las notaciones al valor actual de la variable auxiliar --sea *S* esta suma; que es siempre ≤ 19, como este mismo algoritmo demuestra. Al mismo tiempo podremos ir anotando los dígitos que indicarán el resultado final. Para ello habrá que distinguir dos casos: si la iteración entrega $S \leq 9$, esa suma *S* determina el correspondiente dígito del resultado, y la variable auxiliar obtiene valor 0; en caso contrario, la variable auxiliar obtiene valor 1 y en el resultado debe anotarse *S* - 10. Así, hasta haber procesado los dígitos de la posición λ. Entonces sólo restará *alargar* el resultado, anteponiéndole un dígito correspondiente al valor actual de la variable auxiliar.

Todos conocemos este *algoritmo de suma*, y sabemos que es correcto. Incluso sabemos que permite sumar números *astronómicos, rápidamente*. Porque las sumas *S* de dígitos que debemos efectuar en cada iteración demandan un esfuerzo muy --uniformemente-- acotado, ya que independientemente de los datos, a lo más sumarán 19. Pero sobre todo, porque a lo más deberemos hacer λ veces ese esfuerzo; y λ-1 es menor al logaritmo --base 10-- del mayor de los dos números especificados por los datos. Por eso podemos concluir que existen constantes *a*,*b* --independientes de los datos que especifican la instancia de este problema genérico-- tales que el *tiempo* necesario para producir el resultado final, siempre sea $\leq a+b \cdot \log(\max\{x,x'\})$, donde *x*,*x*' son los valores de los números a sumar.

Ahora, lo crucial es que si bien la función *log* es monótonamente creciente, lo hace muy *lentamente*: números astronómicos --como 10^{15}-- pueden ser anotados usando relativamente pocas --15-- posiciones. Muy diferente sería el mundo de la computación, si para alguna operación básica con números, estuviéramos obligados a iterar procedimientos elementales un número de veces que podría llegar a ser del orden de los números especificados por los datos. Esa operación se tornaría prohibitiva, por muy rápidas que sean las ejecuciones de los procedimientos elementales que requiere cada iteración. Porque ninguna computadora, aunque *opere a la velocidad de la luz*, puede cubrir distancias astronómicas --de años-luz-- en microsegundos. Sólo se podría operar con números *relativamente pequeños*.

Consideremos ahora otro *problema genérico*: Dado un conjunto finito *A* de números naturales, podríamos estar interesados en saber si es posible *biparticionar A* en un par de conjuntos (B,A\B) -- A\B es el *complemento* de *B* en *A*-- tal que $\Sigma(B) = \Sigma(A\backslash B)$ --a saber, que la suma de los números incluidos en *B* resulte igual a la suma de los no incluidos.

Claro, esta es una tarea que incluso hubiera podido interesar a nuestro pastor prehistórico. Por ejemplo, si hubiera poseído varios rebaños --tantos, como números incluye *A*, con poblaciones de ovejas cuyas cardinalidades corresponden a los elementos que componen *A*-- y quisiera heredarlos a sus dos hijos, de tal manera que los rebaños mismos no sean particionados --porque las pobres ovejitas no lo soportarían--, pero que no obstante los dos herederos obtengan la misma cantidad total de ovejas.

Cabe imaginar que en tal caso nuestro pastor sencillamente contará las ovejas de todos sus rebaños, y

que luego probará todas las sumas parciales que corresponderían a las diferentes particiones (B,A\B). Ya sabemos que $B \subseteq A$, dado un particular , su notación posicional le permite *computar* $\Sigma(B)$ muy eficientemente; siempre que $|A|$ --la cardinalidad de *A*, el número de rebaños-- sea *relativamente pequeño*, y por lo tanto también lo sea $|B|$.

Por lo demás, nuestro pastor disponía de tiempo; no era muy frecuente que se le plantearan tales problemas de herencia. Es por eso que probablemente nunca se planteó la cuestión de un *algoritmo* para resolver este tipo de *problema de bipartición*; ni menos, por supuesto, la meta-cuestión de la *existencia de un algoritmo eficiente*.

Hoy, por supuesto, esto último ha cambiado radicalmente: ya no tenemos tanto tiempo, hay muchas ovejas el mundo, y muchísimas otras razones que harían interesante poder disponer de un algoritmo que respondiera a nuestra *cuestión genérica de la bipartición*, eficientemente, para cualquier conjunto *relativamente pequeño* de datos especificantes *A*. Nos referiremos a estas *muchas razones modernas* más adelante. Veamos primero qué nos enseña nuestro *problema de la bipartición*.

Es cierto que para cualquier $B \subseteq A$, $\Sigma(B)$ puede ser determinado eficientemente. Pero para aclarar la cuestión de la *existencia* de un $B \subseteq A$ con $\Sigma(B) = \Sigma(A \backslash B)$, parece que habría que calcular $\Sigma(B)$ para *casi todos* los $B \subseteq A$. Claro, si *hay suerte*, se encontrará pronto un $B \subseteq A$ que satisface la ecuación requerida. ¿Pero si *no hay suerte*; lo que invariablemente será el caso cuando no existe tal *B*? Para constatarlo, ¿habría que probar todos los $2^{|A|}$ subconjuntos de *A*?

Entonces el algoritmo resultaría definitivamente *ineficiente*, puesto que ese número *crece exponencialmente* en función de $|A|$, es decir, se torna *astronómico* para cardinalidades *relativamente pequeñas* de *A*. Por ejemplo, si $|A|=10$, $2^{|A|} \approx 10^3$, si $|A|=30, 2^{|A|} \approx 10^9$ y si $|A|=50, 2^{|A|} \approx 10^{15}$; en este último caso, si una *prueba* demanda un microsegundo de cómputo, el *probar todos los subconjuntos de A*, exigiría computar durante aproximadamente 3 años; y si $|A|=100$, durante 3 mil millones de milenios.

Es por eso que es interesante observar que habrá a lo más $s := \Sigma(A)$ subconjuntos $B \subseteq A$ con sumas S(B) diferentes; y que por lo tanto bastaría saber si entre estas se encuentra una $= \Sigma(A)/2$; y para generar todas esas diferentes *sumas posibles*, se podría usar un *algoritmo de programación dinámica*:

Esto significa, partir con $D := 0$ e ir ampliando paulatinamente este $D \subseteq A$, agregándole en cada iteración un $a \in A \backslash D$, hasta obtener $D = A$. Además, generar en cada iteración el conjunto $\{\Sigma(C);\ C \subseteq D\}$ de las *sumas que se pueden producir* con subconjuntos de *D*. Note que esta *generación* no parece comportar mucho esfuerzo computacional, puesto que si tenemos las *sumas que permiteD* y ampliamos este conjunto, como previsto arriba, agregándole un $a \in A \backslash D$, el conjunto de *sumas permitidas por* $D \cup \{a\}$ es el de *D*, unido al que se obtiene de ese conjunto, sumando a cada una de las *sumas* el número *a*. Es decir que en cada iteración habrá que efectuar a lo más *s* sumas de dos números, por lo tanto en total, a lo más $s \cdot |A|$ tales sumas --y ya sabemos que cada una de estas puede efectuarse muy eficientemente.

Eso no parece prohibitivo. Por ejemplo, si $|A|=100$ y $s=1000$, entonces $s \cdot |A| = 10^5$; en cambio $2^{|A|}$ es más o menos 10^{25} veces la cota que hemos logrado establecer. Sin embargo, este ahorro se da, debido a que *s* fue supuesto relativamente pequeño --supuse que el término medio de las cardinalidades de los rebaños es igual a 10. Porque si este término medio creciera, el número de sumas que puede requerir esta *programación dinámica* crecería de forma proporcional a este término medio. Por lo tanto, este *algoritmo dinámico* tampoco puede ser considerado *eficiente*: porque su *tiempo de cómputo* crece proporcionalmente a las *cardinalidades de los rebaños*, números que crecen *exponencialmente* en función del *largo* de sus notaciones.

Este último argumento, que para el *problema de la bipartición* no parece tan concluyente --porque nos cuesta imaginar números astronómicos de ovejas-- es sin embargo central, para todo lo que quiero decir en este ensayo. Es decisivo para toda la *teoría de la complejidad*, porque si entendiéramos que el visualizado *algoritmo dinámico* es *eficiente*, ello tendría consecuencias teóricas que usted, estimado lector, supongo que ni sospecha:

De hecho se sabe --aunque esto no es algo que yo pueda demostrar aquí-- que un algoritmo eficiente para el *problema de la bipartición*, si estuviera disponible, también sería capaz de aclarar eficientemente las cuestiones que plantean una amplia gama de problemas computacionales; todos los llamados *NP-completos*; clase de problemas que por ejemplo incluye al de la *primacidad de números*, al del *vendedor viajero*, pero también a otros que son puramente combinatoriales --en cuyos planteos no figuran números--, como el de la *consecuencia en lógica proposicional* o el de la *confección de horarios*, y muchos otros de gran relevancia, tanto *teórica* como *práctica*.

Pero lamentablemente todos estos problemas *NP-completos* siguen resistiéndose a un tratamiento computacional eficiente, lo que causa contundentes dificultades *prácticas*. E incluso *teóricas*, porque la existencia de un algoritmo eficiente para cualquiera de los problemas de esta clase conllevaría un *algoritmo universal* para toda la clase. De hecho demostraría algo que resulta definitivamente increíble: que todo *cómputo no-determinista* --que *teóricamente* sería capaz de ejecutar al mismo tiempo, paralelamente, todas las opciones que el programa no-determinista permite-- puede eficientemente ser emulado *determinísticamente* --es decir, en base a un programa determinista que en cada momento permite a lo más una opción. Por todo esto, a estas alturas parece poco sensato soñar con los mencionados *algoritmos eficientes*.

Hay que mencionar, sin embargo, que nadie ha podido demostrar que son imposibles; asunto que no sólo amarga la existencia de muchos matemáticos, sino que, *positivamente*, permite suponer que la distinción *no-exponencial/exponencial*, o *determinado/no-determinado* es fundamental, *filosófica*; que para *matematizarla*, habría que adoptarla como axioma --comparable al *de elección* o a la *hipótesis de Cantor*.

Conclusión

Como sea, lo expuesto articula la gran lección de la *teoría de la complejidad algorítmica*; una lección que es *propiamente matemática*. Porque la *experiencia computacional* bien podría ir enseñando que un determinado algoritmo sirve para ejecutar ciertas tareas eficientemente. Pero mal puede hacer ver que para un determinado problema no cabe esperar algoritmos eficientes; hacer aparecer nítidos límites de la potencialidad computacional.

Para ello se hace necesario *reflexionar sobre* la computación, hacer abstracción de particularidades para relacionar diferentes problemas --y diferentes algoritmos-- en base a características comunes muchas veces muy difíciles de identificar.

Por lo tanto, parece que hemos concluido que las matemáticas, aún si sólo *hablaran* del cómputo, sí logran decir cosas que le debieran interesar a la práctica, porque permiten una mejor *comprensión* de ella.

Pero creo que nuestro *argumento central* también nos permite decir que esta *comprensión matemática* no sólo es una que planea abstractamente sobre los fenómenos computacionales que nos interesan, sino que se entremezcla e involucra directamente con ellos.

Para ello constatemos primero que la *notación posicional (o tamañal)* de las cuentas es *intrínsecamente matemática*, pues debe ser entendida como *comprensión* de la realidad (rebaño) que denota. Ello, porque *está muy lejos* de poder dar lugar a la *extensión* del conjunto de objetos (ovejas) al que se

refiere. No sólo porque abstrae de todas las otras características de esos objetos (ovejas), también porque --de acuerdo a nuestro *argumento central*-- no puede existir *algoritmo eficiente* que dé lugar a un conjunto de objetos cualquiera (por ejemplo, puntos) que tenga la *extensión* denotada: sencillamente porque la magnitud de cualquiera de estos conjuntos *crece exponencialmente* en función del número de dígitos; del *largo* que exhibe la notación (cuenta).

Es extraño que la *teoría de la complejidad algorítmica* que hemos revisado arriba, no haga hincapié en esto último. Tal vez, porque es *matemáticamente* trivial. Pero *filosóficamente* no lo es, ya que demuestra que la notación posicional de cardinalidades de conjuntos no es algo que *representa* al conjunto --en el sentido de *reemplaza* o *trae a la mano* (como diría tal vez Humberto Maturana)-- sino que *comprende y habla* de ese conjunto; es por lo tanto *lenguaje*; aunque un lenguaje especial, *matemático*, con sus sintaxis de guijarros, inscripciones o dígitos (muy liviana); y su semántica precisa, pero sin embargo sin posibilidades de *generar lo significado eficientemente* a partir de la sintaxis sola. En ese sentido es *conocimiento intencional*, y no una mera *plasmación de la extensión* de la realidad articulada.

Son estas las razones que nos llevan a revisar lo planteado en el número anterior de nuestra *Revista* y resumido al comienzo de este ensayo. Son estas las razones que nos llevan a decir que aún si la notación posicional no presupone una noción abstracta de número, porque es *comprensión* y no *representación*, ya es *lenguaje matemático*. Y de ello se desprende que los tipos de datos que el *cómputo* maneja, son *matemáticos*; que las matemáticas no sólo *teoretizan*, sino que participan centralmente y desde siempre en la *práctica* computacional.

De ahí, que la *ecuación* del título de este ensayo resume sus conclusiones. Pero, por supuesto que sobreviven dudas.

¿Si el tipo de datos que maneja el cómputo es de *imágenes* en vez de *números*, sigue dándose un *Cómputo de Matemáticas*, o es que la tan actual *computación gráfica* es radicalmente diferente a la *textual-matemática*? Por ejemplo, para caracterizar un rebaño de ovejas, en vez de indicar su cardinalidad --*número* de ovejas--, podría ser al menos tan interesante presentar una *fotografía* de una de sus ovejas. Al vecino del pastor, a quién este le está ofreciendo sus ovejas, puede que le interese tanto la fotografía de un ejemplar como la cardinalidad de todo el rebaño. Ninguna de las dos caracterizaciones es completa; entre las dos tienden más bien a complementarse; así como típicamente entendemos que se complementan *imágenes* y *palabras*. Por eso se dice que *una imagen reemplaza a mil palabras*, así como una fotografía es imagen de las mil ovejas del rebaño; pero *una palabra resume a mil imágenes*, así como la palabra *mil* resume al rebaño.

Debido a esta complementaridad pareciera que la *computación gráfica* que trabaja con un tipo de datos *imágenes* no es una *computación matemática*, porque ahora lo que estaría siendo *computado* no serían *comprensiones*, sino más bien --ahora sí-- *representaciones* de la realidad.

Incluso podría pensarse que esta es la *auténtica computación*; que la que manipula números y otras comprensiones matemáticas, es una *versión muy particular, matemática*; pero que en general, la computación no es tan *conceptual*, sino mucho más *fáctica*; que no maneja *comprensiones* sino que *representaciones* --que desde el punto de vista de las *teorías* sólo tendrían el carácter de *ejemplos*, como la foto de la oveja lo es en relación a la cardinalidad del rebaño.

Por todo eso es importante preguntarse en qué medida podemos decir que lo que arriba desarrollamos para los números, también es válido para este tipo de datos *imágenes* que parece tan diferente y está tan de moda. Pero hasta aquí nomás por hoy --esto y mucho más, en el próximo capítulo.