Capítulo 0. Introducción al Análisis de Algoritmos

Diego Munguía Molina *

Julio, 2017

Introducción

> *"El problema real es que los programadores han gastado mucho tiempo preocupándose por la eficiencia en los lugares equivocados y en los momentos equivocados; la optimización prematura es la raíz de todos los males (o por lo menos de la mayoría) en programación."* –D. Knuth (Conferencia Premio Turing ACM, 1974)

La eficiencia de los procesos computacionales ha sido un eje fundamental en el desarrollo de las ciencias de la computación. Podemos hipotetizar algunas razones del porqué de nuestro interés por la eficiencia: podríamos decir que es una predisposición natural en ingeniería; además los recursos de hardware, aunque se mantienen en constante evolución, históricamente han sido escasos; y finalmente, no hemos podido solucionar la completitud de todos los problemas computacionales que existen. Con cada avance en software y hardware aparecen nuevos problemas que empujan los límites de la disciplina.

Las primeras computadoras contaban con recursos muy limitados. ENIAC (1943) podía procesar aproximadamente 5,000 operaciones por segundo e inicialmente no tenía memoria, posteriormente fue extendida con una memoria que podía almacenar hasta 100 palabras. La IBM 650 (1953) fue la primer computadora producida en masa. Contaba con una memoria capaz de almacenar entre 1,000 y 4,000 mil palabras. La primera generación del sistema IBM S/360 (1964) podía procesar aproximadamente 29,000 operaciones por segundo y almacenar hasta 64 KB de datos en memoria. Hasta este punto la computación era una actividad tecnológica que sólo podía llevarse a cabo en universidades, grandes corporaciones y principalmente complejos militares; las computadoras eran grandes *mainframes* que ocupaban salones completos. En 1968 llega a Costa Rica "Matilde", la primer computadora para aplicaciones científicas del país, una *mainframe* IBM 1620 adquirida

por la Universidad de Costa Rica. En la década de 1970 surgen los primeros microprocesadores y con ellos las primeras computadoras personales. El procesador Intel 8080 (1974) es el ancestro directo de los procesadores actuales de arquitectura `x86`. Corría con una frecuencia de reloj de 2 MHz y permitía direccionar hasta un máximo de 64 KB de memoria.

Con estas limitaciones, los investigadores en ciencias de la computación, ingenieros de sistemas computacionales e ingenieros de software se han visto obligados a pensar creativamente en sus soluciones, aprovechándose del hecho de que usualmente existe más de una forma de resolver un mismo problema computacional. La meta desde ese entonces ha sido resolver el problema utilizando la menor cantidad de recursos computacionales posibles.

Pero qué pasa hoy en día que contamos con procesadores de múltiples núcleos, que corren con frecuencias de reloj de 3 GHz o más, y con capacidades de memoria y almacenamique se pueden medir en terabytes. ¿Sigue siendo la eficiencia un problema que nos concierne? Una posible respuesta a esta pregunta podría ser "hoy más que nunca". La tecnología computacional hoy en día está más presente que nunca en nuestras sociedades, principalmente en forma de dispositivos móviles. Podemos encontrarla en poblaciones urbanas y rurales, y en diversos estratos culturales y socioeconómicos. Atrás han quedado los tiempos en que la computación era asunto exclusivo de militares, empresarios y académicos. No sólo están estos dispositivos presentes en vastas extensiones de las sociedades humanas, sino que están además conectados entre sí a través de internet. Un efecto de esta ubicuidad computacional es la producción de masivas cantidades de datos que pueden ser procesados, llevando nuevamente al límite problemas clásicos que se pensaban resueltos para todo efecto práctico.

Problemas computacionales y algoritmos

Antes de hablar más en detalle sobre eficiencia, debemos detenernos primero en el concepto de algoritmo, pues es a través de algoritmos bien diseñados que llegaremos a procesos computacionales eficientes.

Podemos adelantar que los algoritmos son soluciones a problemas computacionales.

Problema Computacional Establece una relación entre la especificación general de una entrada que describe todas las posibles instancias del problema y la especificación de la salida esperada.

(1) **Ejemplo**. Suma binaria.

Entrada. Una tupla que contiene dos hileras, cada hilera representa un número binario correctamente formado.

Salida. Una hilera que representa el número binario resultado de sumar los dos números de entrada.

Es importante notar que el problema sigue la noción de caja negra, sólamente especificamos cuáles son las entradas y qué forma tienen, y cuáles son las salidas y qué forma tienen. No se indica cómo se soluciona el problema, es decir cómo se producen las salidas a partir de las entradas o cómo se transforman las entradas en las salidas.

El problema debe tener cierta generalidad. La especificación de la entrada debe ser lo suficientemente general para tomar en cuenta todos los posibles casos (**instancias**) que nos interese resolver. Un problema no general podría leerse como *"toma un 5 y un 2 como entrada y produce un 7 como salida"*. Una versión más general podría ser *"toma dos números naturales a y b y produce como salida un número natural que corresponde a la suma $a + b$"*.

Cabe resaltar que en este punto no nos preocupa ningún tipo de restricción impuesta por el modelo de computación que se utilizará para resolver este problema. Existen múltiples posibilidades de máquinas físicas y teóricas que podríamos utilizar para intentar resolver los problemas dependiendo de nuestros objetivos; no es necesario limitar el problema a sólo una de ellas.

Los problemas computacionales son producto de la abstracción; por esta razón a estos problemas también se les conoce como **problemas abstractos**. Recordemos que el proceso de abstraer implica aislar propiedades de objetos concretos para después razonar sobre ellas. En otras palabras, podemos percibir y pensar en muchos detalles acerca de una situación particular de la realidad; cuando abstraemos esta situación ignoramos algunos detalles y prestamos atención a otros, en función del objetivo que queremos lograr.

Podemos pensar por ejemplo en una baraja de cartas o naipes, si observamos una baraja francesa particular podríamos enumerar una serie no exhaustiva de características:

- Cada naipe es de cartón.
- La baraja tiene 52 naipes.
- Cada naipe mide 8.9×6.4 cm.
- Cada naipe tiene el mismo diseño gráfico al reverso.
- La baraja está compuesta por cuatro grupos llamados palos de 13 naipes cada uno.
- Cada palo está identificado por un símbolo: espadas ♠, tréboles ♣, diamantes ♢ y corazones ♡.
- Cada palo consta de 13 naipes: 9 naipes numerados del 2 al 10 y 4 naipes marcados con las letras A (as), J (paje), Q (reina) y K (rey).
- Los naipes del palo de las espadas o tréboles están impresos en tinta negra.
- Los naipes del palo de los corazones o diamantes están impresos en tinta roja.

Supongamos que queremos modelar el juego de *veintiuno* o *blackjack*. Este problema nos provee un marco de referencia para abstraer detalles de nuestro conjunto de observaciones que son importantes dentro del contexto del juego.

En nuestra versión simplificada de veintiuno, el jugador se enfrenta a la casa. Ambos reciben inicialmente dos naipes. El jugador y la casa pueden solicitar más naipes. El objetivo de cada parte es maximizar la suma de los valores de las cartas. Gana quien tenga la mayor suma, siempre y cuando esta no exceda 21. Los naipes J, Q y K tienen un valor de 10, y A puede tener un valor de 1 u 11.

Para abstraer este juego no nos interesa el material de los naipes, ni sus dimensiones, colores o gráficas. Sólo necesitamos su valor numérico. Además nos podría interesar a cuál palo pertenece cada naipe, pues de esta forma podemos validar que las manos sean válidas, por ejemplo si jugamos con un sólo mazo, no puede haber más de cuatro naipes con valor 8.

Una vez determinadas las características relevantes para el problema, podemos proceder a definir la forma en que se presentarán los objetos involucrados. Podemos representar un naipe como una tupla que contiene el valor de la carta y el símbolo del palo al que pertenece, y podemos representar una mano como un conjunto de naipes – es decir de tuplas `(valor, símbolo)`.

A la hora de expresar el problema, podemos utilizar prosa para explicar los detalles, restricciones y formas de las entradas y salidas. Opcionalmente podemos hacer uso del lenguaje de la lógica, el cual permite que las expresiones sean breves, concisas y precisas.

Utilizando lógica, podemos definir una baraja como el producto cartesiano entre el conjunto de valores y el conjunto de símbolos.

$$Baraja = \{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\} \times \{\spadesuit, \clubsuit, \diamondsuit, \heartsuit\} \tag{1}$$

Y con esta definición podemos plantear el problema computacional de determinar cuál mano es la ganadora de acuerdo con nuestras reglas simplificadas del juego de veintiuno:

(2) **Ejemplo**. Veintiuno.
Entrada. $(ManoJ, ManoC)$ $t.q.$ $ManoJ \subset Baraja \land ManoC \subset Baraja \land ManoJ \cap ManoC = \emptyset$
Salida. 1 si ManoJ gana a ManoC, 0 en el caso contrario.

Una **instancia** de un problema computacional se refiere a la expresión de una entrada y una salida particular que cumplen con las restricciones del problema general abstracto.

(3) **Ejemplo**. Instancia de veintiuno.

Entrada. $(\{(A, \spadesuit), (Q, \heartsuit)\}, \{(10, \diamondsuit), (8, \clubsuit), (2, \heartsuit)\})$

Salida. 1

Algoritmo Especificación de la solución a un problema computacional.

El algoritmo especifica cómo se transforman las entradas en salidas. La transformación sucede a través de una secuencia discreta de pasos de un estado a otro. Esta secuencia puede ser explícita o tácita.

Más allá de esta idea, no hay un consenso sobre una definición exhaustiva del concepto de algoritmo. A pesar de esto, es posible enumerar una serie de características deseables en un algoritmo:

- Desde una perspectiva teórica, el algoritmo debe ser *correcto* o *eficaz*, es decir debe resolver el problema general.
- Desde una perspectiva práctica, el algoritmo debe ser *eficiente*, es decir debe llegar a una solución utilizando la menor cantidad de recursos posibles.

Eficiencia

Al considerar la eficiencia de los algoritmos entramos en el campo de la ingeniería y la aplicación práctica y por tanto debemos considerar la posibilidad de ejecutar los algoritmos en máquinas físicas.

Los algoritmos consumen recursos de las máquinas que los ejecutan, principalmente *ciclos de procesador* y *memoria*, pero también almacenamiento persistente, interfaces de red, memorias caché y GPUs, entre otros.

Los algoritmos eficientes minimizan sus requerimientos de recursos.

Los principales aspectos de eficiencia a considerar en un algoritmo son *tiempo* y *espacio* correspondientes a CPU y memoria respectivamente.

La experiencia nos muestra que es común incrementar el consumo de memoria cuando se trata de minimizar el tiempo de ejecución de un algoritmo; respectivamente al minimizar el consumo de memoria es común requerir de más ciclos de procesador. Esta observación no es una regla invariable pero si un patrón que podemos observar frecuentemente.

En los siguientes capítulos se presentarán algunas herramientas lógicas y matemáticas que nos permitirán razonar formal y sistemáticamente sobre el consumo de recursos en los algoritmos.

Esto nos permitirá hacer observaciones y análisis sobre el diseño de algoritmos y conocer sobre los límites

del campo de la computación.