Introducción

En entradas anteriores ya hablamos de problemas algorítmicos y de algoritmos para resolverlos. También ya discutimos de la importancia de que los algoritmos sean correctos, y vimos cómo podemos determinar si un algoritmo es correcto o incorrecto.

Ahora nos enfocaremos en otra de las propiedades deseables de los algoritmos: que sean eficientes. Veremos cómo vamos a comparar distintos algoritmos entre sí. Hablaremos de cómo contar cuántos pasos toman de manera precisa. Después, hablaremos de por qué basta con encontrar una cuenta aproximada.

Modelo RAM

Puede haber más de un algoritmo que resuelva un mismo problema algorítmico. Nos interesa poder comparar distintos algoritmos para saber cuál de ellos elegir. Una forma de hacer esto es considerar cuánto tiempo tarda un algoritmo en resolver el problema.

Para esto, tenemos que ponernos de acuerdo en cómo medir este tiempo. Esta forma de hacerlo debe de ser teórica y no depender de si tenemos una computadora o un lenguaje de programación más rápido. Es por esto que debemos fijar un **modelo** computacional, en donde debemos hacer algunas suposiciones de cuánto tiempo tardan ciertas operaciones.

Un modelo que funciona bastante bien y modela con mucha precisión cómo funcionan las computadoras actualmente es el **modelo RAM**.

En este modelo, las «operaciones sencillas» toman exactamente un paso. Algunos ejemplos de operaciones sencillas son las siguientes:

- Leer un objeto pequeño de la memoria (cache o disco duro).
- Sumar, restar, multiplicar, dividir dos números.
- Dados dos números, decidir cuál es más grande o si son iguales.
- Mostrar al usuario un mensaje corto.

Las siguientes son ejemplos de operaciones que, de entrada, no son sencillas

- Correr todas las ejecuciones de un bucle.
- Calcular la suma de todos los elementos de un arreglo.
- Multiplicar varios números.

Dejando claras cuáles son las operaciones sencillas, es posible decir exactamente cuántos pasos toma un algoritmo.

Ejemplo. Un algoritmo que realice la suma de todos los elementos de la lista $[1,6,4,2,7]$ y verifique si esta suma es igual a 27, podría proceder haciendo las siguientes operaciones del modelo RAM:

- Una operación para iniciar una variable S en 0.
- Una operación para leer 1, una para sumarlo a S y otra para almacenar este valor en S.
- Una operación para leer 6, una para sumarlo a S y otra para almacenar este valor en S.
- Una operación para leer 4, una para sumarlo a S y otra para almacenar este valor en S.
- Una operación para leer 2, una para sumarlo a S y otra para almacenar este valor en S.
- Una operación para leer 7, una para sumarlo a S y otra para almacenar este valor en S.
- Una operación para reportar S.

En total tendríamos $1+3+3+3+3+3+1=17$ operaciones. Este es un problema muy sencillo, pero deja ver una desventaja de usar

En total tendríamos $1+5+5+5+5+1=17$ operaciones. Este es un problema muy sencillo, pero deja ver una desventaja de usar únicamente el modelo RAM: típicamente un algoritmo necesita muchas operaciones y puede ser algo difícil mantener una cuenta exacta. Más adelante hablaremos de cómo superar esta dificultad. $\square$

Tipos de complejidades

El modelo RAM nos permite saber exactamente para un problema dado, un algoritmo dado y una instancia dada, cuántos pasos tardará el algoritmo en resolver el problema. Sin embargo, suena injusto comparar algoritmos de acuerdo a lo que pasa únicamente en una (o unas pocas) de sus instancias. En general, nos gustaría comparar algoritmos entre sí, pero a través de todas las posibles instancias que se deben resolver.

Usualmente las instancias tienen cierto tamaño n. Por ejemplo, podemos querer ordenar n objetos. O sumar n números. O buscar un objeto en una lista con n objetos ordenados. Una muy buena forma de evaluar un algoritmo es contar cuánto tiempo tarda el algoritmo en evaluar las instancias de tamaño n. Hay por lo menos 3 formas de hacer esto:

- Con la **complejidad del mejor caso**, es decir, para cada valor de n, fijarnos en aquella instancia de tamaño n que le tome al algoritmo la menor cantidad de pasos, y llamar a este número $m(n)$.
- Con la **complejidad promedio**, es decir, para cada valor de n, hacemos el promedio $p(n)$ de pasos que le toma al algoritmo, tomado sobre todas las posibles instancias de tamaño n.
- Con la **complejidad del peor caso**, es decir, para cada valor de n nos fijamos de entre todas las instancias de tamaño n aquella que le toma más pasos al algoritmo y a esta cantidad le llamamos $M(n)$.

Considerar cada una de estas complejidades tiene sus ventajas y desventajas.

La complejidad $m(n)$ del mejor caso es muy optimista. Nos permite saber qué es lo mejor que puede hacer el algoritmo «si tenemos suerte». Es como cuando compramos un boleto de lotería y pensamos «en el mejor de los casos, me llevaré el premio mayor».

La complejidad promedio $p(n)$ es quizás la más justa, pues no descarta los tiempos buenos o malos del algoritmo, sino que promedia todo. Es como cuando compramos un boleto de lotería y pensamos probabilísticamente en la cantidad esperada de dinero que ganaremos. Tiene la desventaja de que, de entrada, habría que calcular todos los posibles tiempos sobre todas las posibles instancias, y esto puede ser muy difícil.

La complejidad $M(n)$ del peor caso es pesimista, pero tiene una gran ventaja. Si logramos encontrar (o entender)$M(n)$, esto nos da una garantía de que esto es lo peor que puede pasar, y entonces en cualquier instancia tendremos que el algoritmo toma esa cantidad de pasos, o menos. Es como dar una cota superior para el tiempo de ejecución. En términos de lotería, equivale a pensar que «lo peor que me puede pasar es no ganar nada y haber perdido lo del boleto de lotería». Como lo peor es haber perdido el costo de un boleto, quizás esto nos anima a sí comprarlo. Otra ventaja de $M(n)$ es que usualmente no es tan difícil de calcular o acotar superiormente.

Pensamiento asintótico

En muchas ocasiones de la vida cotidiana nos interesa contar cierta cantidad, pero únicamente de una manera aproximada. Contar de manera exacta la cantidad probablemente nos cueste mucho trabajo y es posible que baste una aproximación para poder tomar decisiones. Un pequeño ejemplo de esto es decidir hacer un viaje de acuerdo a cuánto costará. Usualmente no nos interesa saber cuántos pesos y centavos costará exactamente. Basta con estimar quizás con la precisión de cientos o miles de pesos.

Cuando estamos estudiando algoritmos sucede algo similar. El modelo RAM nos permitiría, en teoría, saber exactamente cuántos pasos toman nuestros algoritmos. Pero usualmente encontrar una cota exacta es demasiado difícil: hay muchas operaciones que se ejecutan, muchas llamadas a memoria, varios bucles, etc. Sería muy fácil perder la cuenta exacta. Y muchas veces ni siquiera nos importa la cuenta exacta, sino simplemente un esimado.

Así como en el ejemplo del viaje sólo nos interesa la precisión a cientos de pesos, o miles de pesos, en la teoría de complejidad computacional sólo nos interesa un **estimado asintótico** de la cantidad de pasos que tomaremos. En la siguiente entrada diremos con precisión matemática a qué nos referimos. Pero por ahora, veamos solamente unos cuantos ejemplos intuitivos de a qué nos referimos

precisión matemática a que nos referimos. Pero por ahora, veamos solamente unos cuantos ejemplos intuitivos de a qué nos referimos.

Problema. Aproximadamente, ¿cuántas palabras están escritas en esta entrada?

Solución. La primera línea de la entrada tiene alrededor de 20 palabras. La entrada tiene aproximadamente 100 líneas. Entonces un estimado decente es que en total la entrada tiene $20\times 100=2000$ palabras. El orden de magnitud es correcto: en efecto estamos en los «miles» de palabras. Puedes convencerte fácilmente de que no estamos en los «cientos» ni en los «diez miles», y después verificarlo con un contador de palabras. $\square$

Problema. Aproximadamente, ¿cuántos círculos de radio 1 caben en un cuadrado de lado 100 sin que se traslapen? ¿Y en uno de lado n?

Solución. Daremos únicamente estimados para la respuesta, pues no nos interesa dar el número exacto. Es fácil ver que siempre podemos poner 2500 círculos dentro de un cuadrado de lado 100 sin que se traslapen. Para hacerlo, dividimos al cuadrado en una cuadrícula de 50×50 hecha por cuadrados de 2×2. Dentro de cada uno de estos cabe un círculo de radio 1 (diámetro 2). Por otro lado, es fácil acotar la cantidad de círculos con un argumento de áreas. Cada círculo de radio 1 tiene área π. El cuadrado de lado 100 tiene área 10000. Como $10000/\pi\approx 3183.09$, entonces no pueden caber más de 3184 círculos.

Llamemos $S(n)$ a la cantidad de círculos de radio 1 que pueden acomodarse dentro de un cuadrado de lado n. El argumento de dividir en cuadrados de 2×2 muestra que:

$$S(n)\geq \left \lfloor \frac{n}{2} \right \rfloor ^2 \approx \frac{n^2}{4}$$
y el argumento de áreas muestra que

$$S(n)\leq \frac{n^2}{\pi}.$$

Aunque no sepamos exactamente cuántos círculos caben, por lo menos estas cotas nos dicen que la respuesta «es cuadrática en n». Esto no está nada mal. Dependiendo de para qué nos estamos haciendo esta pregunta, es posible que estas cotas sean más que suficientes. $\square$

Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Realiza estimaciones para las siguientes cantidades. Si vas a averiguar cifras en internet, averigua lo menos posible.
    - La cantidad de segundos que dura una lista de reproducción de la radio de 50 éxitos latinos.
    - La cantidad de caracteres en esta libreta de Jupyter.
    - Los pasos que hay que dar para recorrer la Ciudad de México de su punto más al norte, a su punto más al sur, suponiendo que se puede en línea recta.
    - La suma de las cantidades de amigos de tus amigos.
    - La cantidad de galletas que debes comprar para los coffee-breaks de un congreso de una semana al que asistirán unos 500 investigadores, y cuánto te gastarás en galletas.
    - La cantidad de MB que necesitarás para grabar todo un día de ruido en el metro.
2. Realiza un estimado por abajo y uno por arriba de cuál es la máxima cantidad de caballos de ajedrez que puedes poner en un tablero de $n\times n$ de modo que no se ataquen entre sí. Intenta que tus estimados queden lo más cerca posible entre sí.
3. Se tiene un cuadrado de 8×8. Se van a pintar sus casillas en blanco o negro de manera aleatoria e independiente. Diremos que un par de casillas son **malas** si comparten un lado en común y además quedaron del mismo color. ¿Cuál es la mayor cantidad posible de pares de casillas malas que pueden aparecer? ¿Y la menor cantidad? ¿Y la cantidad promedio?
4. Imagina que quieres multiplicar dos matrices de 2×2. Se valen hacer sumas (de dos en dos elementos) o multiplicaciones (de dos en dos elementos). Tienes que pagar 1 peso cada vez que sumes dos números y 2 pesos cada vez que multipliques

dos números. ¿Cuántos pesos tienes que pagar en total? ¿Y si fueran matrices de \(n\times n\)?

5. Investiga de manera general qué otros modelos computacionales existen.

‹
Anterior
Algoritmos correctos

Siguiente
Notación O grande y similares
›

Por Leonardo Ignacio Martínez Sandoval