Trabajo de Investigación para el Curso BMAT-01 (borrador)

Aplicaciones del Cálculo en la Ingenieria de Software: Optimización de Algoritmos y Métodos Criptográficos

Francisco Díaz, John García Calvo, Roberto Vindas Hernández, Weslin Mena Montero

UNIVERSIDAD LATINA DE COSTA RICA
LAUREATE INTERNATIONAL UNIVERSITIES®

19-07-2019

Abstracción

Una explicación de la importancia que los conocimientos matemáticos juegan en la ingeniería de software. Utilizando ejemplos simplificados para hacer los conceptos accesibles a gente no familiarizada con el campo.

Introducción

El poder de los computadores ha crecido de forma exponencial en las ultimas décadas, (un celular de hoy, es significativamente más avanzado que el ordenador a bordo del Apollo 11) sin embargo no siempre se explota su máximo potencial.

Una de las características más fundamentales de las computadoras es que estas solo pueden ser tan eficientes como el software que corren, por ende el software ineficiente, será ineficiente sin importar que tan sofisticado o moderno sea el Hardware donde corre.

Como ingenieros es aquí donde los conocimientos adquiridos en el curso de “Cálculo I” (BMAT-01) resultan altamente útiles para poder por ejemplo:

- Analizar los **limites** de dos piezas de software para determinar cual será más rápido
- Utilizar **derivación** para poder saber cual será el tiempo de ejecución de un algoritmo en el peor y mejor de los casos.

Además hay procesos informáticos de los cuales dependemos diariamente y (damos por sentado) como la “encripción RSA” que mantiene nuestros datos personales, y contraseñas protegidas y en secreto. Este proceso no sería posible de implementar de no tener un fundamento fuerte matemático por ejemplo:

- La definición del teorema de números primos se puede probar buscando un **limite**
- Para probar la seguridad de este algoritmo utilizamos una **integral**

Justificación del Tema

Existe mucha gente incluso dentro de las otras áreas de ingeniera que no esta familiarizada con lo que hacen los ingenieros de software.

Pretendemos poder explicar brevemente y poner en contexto a estos grupos explicando

- Que significa programar

- Que es un algoritmo
- Cuales son los retos a la hora de escribir buen algoritmo?

Existen también ingenieros de software que no tienen muy clara la conexión que existe entre su área y la matemática analítica, pretendemos recordarle a estos:

- Como pueden empezar a usar el cálculo para mejorar el software que escriben
- Motivar a los estudiantes a considerar utilizar un enfoque matemático a la hora de investigar temas de software y no únicamente la parte más técnica de lenguajes de programación

I Parte - Optimización de Algoritmos

Medidas de Eficiencia

Objetivo:

- **Describir de manera exacta pero simplificada el desempeño de un Algoritmo.**

La especificación exacta del tiempo de ejecución de un algoritmo esta defininda por una operación de estructuras discretas

- Conjunto I de las Instancias
- Tiempo de Ejecución del Algoritmo: $T : I \to \mathbb{N}$

Problema:

T **es muy difícil de determinar y describir**

Solución

Agrupación de las instancias (por tamaño)

Dado el conjunto de Instancias I_n de **tamaño n** de un problema, podemos medir la eficiencia y determinar:

Worst case:

Nos garantiza la eficiencia del algoritmo, pero posiblemente con una aproximación demasiado pesimista

$$t(n) = max\{T(i) : i \in I_n\}$$

Average case:

Nos da el tiempo de ejecución promedio, pero este no corresponde necesariamente al tiempo real en la práctica

$$t(n) = \frac{1}{|I_n|}\sum_{i \in I_n} T(i)$$

Best case:

Comparable al worst case, nos da una aproximación posiblemente demasiado optimista

$$t(n) = min\{T(i) : i \in I_n\}$$

Nos aseguramos que I sea finito y que el minimo y máximo elemento existan

Las ecuaciones exactas para t(n) son demasiado complejas sino imposibles para esto utilizamos:

Crecimiento asintótico

$f(n)$ y $g(n)$ tienen el **mismo índice de crecimiento**, si la proporción esta delimitada por constantes c y d

$$\exists c, d \in \mathbb{R}_+ \ \exists n_0 \in \mathbb{N} \ \ \forall n \geq n_0 : c \leq \frac{f(n)}{g(n)} \leq d$$

$f(n)$ crece **más rápido** que $g(n)$ cuando exista un n_0 para todos las constantes positivas c , a partir de $f(n) \geq c \cdot g(n)$ para $n \geq n_0$

$$\forall c \in \mathbb{R}_+ \ \exists n_0 \in \mathbb{N} \ \ \forall n \geq n_0 : f(n) \geq c \cdot g(n)$$

O en otras palabras

$$\lim_{x\to\infty} \frac{g(n)}{f(n)} = 0$$

Conjuntos para la formalización de la notación asintótico

$O(f(n)) : \{g(n) : \exists c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \leq c \cdot f(n))\}$

$\Omega(f(n)) : \{g(n) : \exists c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \geq c \cdot f(n))\}$

$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

$o(f(n)) : \{g(n) : \forall c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \leq c \cdot f(n))\}$

$\omega(f(n)) : \{g(n) : \forall c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \geq c \cdot f(n))\}$

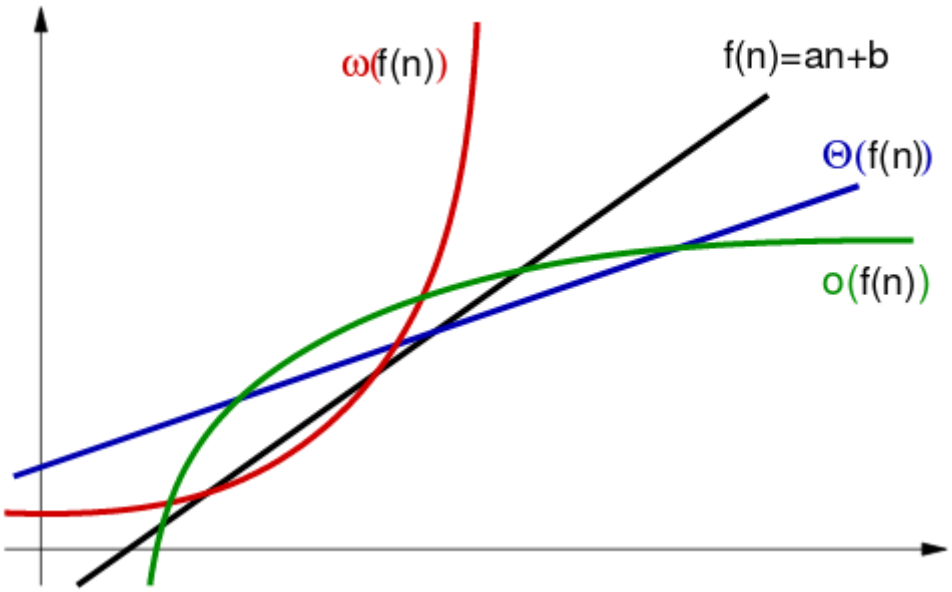

Figure 1: Gráfica

Basándonos en esta información podemos determinar dentro de cual con conjunto cae nuestro algoritmo

Ejemplos:

$$5n^2 - 7n \in O(n^2)$$

$$5n^2 - 7n \in \Omega(n^2)$$

$$n^2/10 + 100n \in O(n^2)$$

$$\log n \in o(n)$$

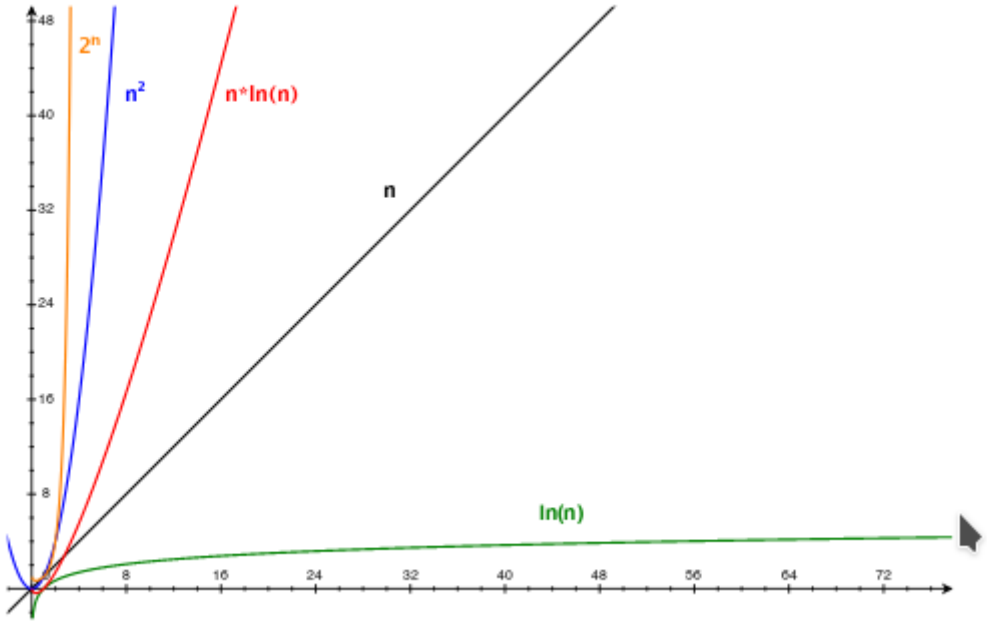

Figure 2: Crecimientos habituales de Algoritmos

Crecimiento ilustrado

> Utilizando un Ordenador con frequencia de reloj de 1GHZ

> Una operación requiere 1ns (nano segundo)

> n representa la cantidad de operaciones

n	$T(n) = \ln(n)$	$T(n) = n$	$T(n) = n\ln(n)$	$T(n) = n^2$	$T(n) = 2^n$	$T(n) = n!$
10	$0.003\mu s$	$0.01\mu s$	$0.033\mu s$	$0.1\mu s$	$1\mu s$	3.63ms
20	$0.004\mu s$	$0.02\mu s$	$0.086\mu s$	$0.4\mu s$	1ms	77.1 years
30	$0.005\mu s$	$0.03\mu s$	$0.147\mu s$	$0.9\mu s$	1s	$8.4 \cdot 10^{15}$ye
40	$0.005\mu s$	$0.04\mu s$	$0.213\mu s$	$1.6\mu s$	18.3min	
50	$0.006\mu s$	$0.05\mu s$	$0.282\mu s$	$2.5\mu s$	13 days	
100	$0.007\mu s$	$0.1\mu s$	$0.644\mu s$	$10\mu s$	$4 \cdot 10^{13}$years	
1000	$0.010\mu s$	$1.0\mu s$	$9.966\mu s$	1ms		
10000	$0.013\mu s$	$10\mu s$	$130\mu s$	100ms		
100000	$0.017\mu s$	0.1ms	1.67ms	10s		
$1 \cdot 10^6$	$0.020\mu s$	1ms	19.93ms	16.7min		
$1 \cdot 10^7$	$0.023\mu s$	0.01s	0.23s	1.16 days		
$1 \cdot 10^8$	$0.027\mu s$	0.1s	2.66s	115.7 days		
$1 \cdot 10^9$	$0.030\mu s$	1s	29.9s	31.7 years		

Figure 3: Tabla de Tiempo de Procesamiento

Derivadas de la Notación "O"

Lema

Si f y g son diferenciables

Entonces:

$$Si\ f'(n) \in O(g'(n)),\ entonces\ f(n) \in O(g(n))$$

$$Si\ f'(n) \in \Omega(g'(n)),\ entonces\ f(n) \in \Omega(g(n))$$

$$Si\ f'(n) \in O(g'(n)),\ entonces\ f(n) \in O(g(n))$$

$$Si\ f'(n) \in o(g'(n)),\ entonces\ f(n) \in o(g(n))$$

$$Si\ f'(n) \in \omega(g'(n)),\ entonces\ f(n) \in \omega(g(n))$$

** Sin embargo lo inverso no siempre aplica **

Ejemplo:

1. Lema:

$$n^3 - 3n^2 + 2n \in O(n^3)$$

$$O(\sum_{i=1}^{n} i) = O(n^2/2 + n/2) = O(n^2)$$

2. Lema:

$$De\ log\ n \in O(n)\ sabemos\ que\ O(n^2)$$

3. Lema:

$$(\log n)' = 1/n, (n)' = 1\ y\ 1/n \in O(1) \implies \log n \in O(n)$$

II Parte - Métodos Criptográficos

Atentos a la situación, una persona A, llamemosla Alice quiere mandarle un mensaje a una persona B, dígamosle Bob, y el mensaje bueno…es un poco personal, y pasa que Alice no se fía y piensa que alguien puede estar espiándolos, y efectivamente a medio camino está Eva.

Si Alice manda algo a Bob, Eva lo va a interceptar y su privacidad no estará a salvo. ¿Que puede hacer Alice para mantener en secreto su mensaje? Es necesario que Alice lo modifique para que solo Bob y ella puedan entenderlo, es decir, tiene que encriptarlo.

Hay muchas formas de hacerlo, desde desplazar las letras en el alfabeto (Cifrado Cesar), hasta incluso usar física cuántica, pero esta vez vamos a hablar de uno de los más seguros y usados.

El método RSA

Con este, si Alice quiere mandarle algo a Bob, Bob le da una caja para la cual sólo él tiene la llave, ella mete su mensaje ahí y se lo manda. Esta caja es lo que se conoce como una clave pública, mientras que la llave es una clave privada.

¿Cómo es que se crean estas claves?

1. Tomar 2 números primos

> *En este ejemplo van a ser 5 y 11* pero normalmente suelen usarse números de cientos de dígitos de longitud. Nosotros diremos que:

$$p = 5\ y\ q = 11$$

2. Multiplicarlos y al resultado lo llamarlo “n”,

> En este caso, vale n = **55**, porque $5 \cdot 11 = 55$.

3. Averiguar cuántos números enteros positivos menores o iguales a “n” son coprimos con este, es decir en los que el máximo común divisor es 1.

> Para eso usamos la función fi de Euler, la cual dice que tenemos que restarle 1 a “p” y a “q” y multiplicar los resultados entre sí. En este ejemplo:

$$fi = (p-1) \cdot (q-1) = 4 \cdot 10 = 40$$

4. Conseguir un número entero menor a “fi” que sea coprimo con este

> Por ejemplo en nuestro caso

$$e = 7$$

5. Crear las claves

Crear clave la pública

> Esta se forma con “n” y “e” colocando esos números así: (n,e) que en este caso sería:

$$(55, 7)$$

Crear la clave privada

> Para eso usamos la fórmula:

$$d = frac1 + k \cdot fie$$

> Donde $k \in \mathbb{Z}$ número cuyo valor aumenta en 1 con cada ciclo, es decir, que primero vale 1, luego

> 2, después 3 etc va hasta que el resultado sea un entero también. En este caso, el valor de k debe ser 4:

$$k = 4 \implies d = \frac{1 + 4 * 40}{7} = \frac{161}{7} = 23\, y\, 23 \in \mathbb{Z}$$

> Finalmente, para formarla, usamos “n” y “d”, colocando los números así: (n,d)

Estas van a ser las claves de Bob: **pública=(55,7) privada=(55,23)**

Ahora regresemos al ejemplo del principio, Alice quiere enviarle algo a Bob, una “c”, por ejemplo.

Primero toma esa “c” y le asigna un número, por comodidad 3, que es su posición en el alfabeto.

Luego toma ese número y lo usa para aplicar la siguiente fórmula

$$a = c^e \mod n$$

Este valor encriptado es básicamente el mensaje de Alice luego de meterlo a la caja que Bob le dió.

Luego, si Bob quiere recuperar el mensaje original, simplemente tiene que usar esta otra fórmula $b = a^d \mod n$, con lo que va a conseguir el 3 que Alice le quería enviar.

¿Pero no es fácil para una computadora simplemente probar un montón de números hasta encontrar el que es y así conseguir la información?

Pues resulta que no, ya que para eso tendría que descifrar “p” y “q”,números que normalmente suelen tener 617 dígitos cada uno.

Cada número tiene 10617combinaciones posibles, y considerando que son dos números con la misma cantidad de dígitos se puede deducir que 106172=101234es la cantidad de combinaciones posibles, un número absurdamente grande incluso para una computadora, y lo peor de todo…es que solo una de esas combinaciones es la correcta.

¿Pero qué tal si Eva consigue una computadora que sea capaz de descifrar esos números rápidamente?

Con incrementar la cantidad de dígitos de cada número a los 1233, las combinaciones subirían a los 102466, lo cual es miles de millones de veces más difícil de descifrar…y puede que me esté quedando corto.

No importa qué tan buena sea la computadora de Eva, siempre habrán números más grandes que se puedan usar. Y eso es algo completamente comprobable al resolver esta integral

$$Li(x) = \int_2^x \frac{dt}{\ln t}$$

cuyos pasos están disponibles en el siguiente link