Apuntes Estructuras de Datos Avanzadas

APUNTE NO. 1 Lunes 25 de enero

Introducción.

Para que un programa sea eficiente hay que verificar el manejo de sus recursos: tiempo [no. pasos], espacio [memoria] y comunicación [no. mensajes].

El tiempo es un problema, es relativo al hardware. Porque podemos tener un mismo programa que se tarda más en una computadora con menos RAM que otra. Entonces ¿qué podemos utilizar? Es como un proxy del tiempo: el número de pasos o de instrucciones. Aún así tenemos el problema de la relación entre el número de pasos y el tamaño de la entrada. Dependiendo del programa vamos a tener comportamientos lineales, exponenciales o logarítmicos. Esto es lo que nos importa: el "crecimiento" del sistema en cuanto a la resolución del problema, es decir, su comportamiento en relación con la entrada. Puede tener un comportamiento constante [el número de pasos no cambia independientemente del tamaño de entrada (inclusive podrían no tener entrada)], pero pocos programas son así. Queremos ver cómo cambia el número de pasos conforme al incremento del tamaño de la entrada. Este comportamiento es el que nos importa para determinar la eficiencia de un programa. Fijarnos en entradas particulares no tendría ningún sentido, sino más bien cómo cambia.

Para hacer el análisis de la eficiencia lo tenemos que hacer contingente al tamaño de la entrada. Para muchas cosas la tecnología no nos va a salvar. Por ello, es importante analizar la complejidad de los programas

APUNTE NO. 2 Miércoles 27 de enero

Como lo que nos interesa es el comportamiento asintótico, vamos a deshacernos de algunas cosas que no influyen en ese comportamiento, que no valen la pena. Por ejemplo, la inicialización de las variables. Las constantes no van a afectar tanto el comportamiento asintótico, solo van a cambiar dónde empieza la curva. Me voy a fijar en las variables que dependen de la entrada y de ciclos: for, while y llamadas recursivas. Nos vamos a fijar en el número de ciclos anidados que tiene un programa, pero no es suficiente verlo de esa manera, es también verlos que dependan del tamaño de la entrada. Por ejemplo, podemos tener un for que ejecuta un for m veces dependiendo de una entrada n; lo que importa realmente es el for que depende del tamaño de la entrada, porque el segundo for se va a ejecutar n veces, es decir, es lineal.

Vamos a checar primero esto.

$$O(g(n)) = \{ f(n): \ \exists\, c, n_0 > 0 \quad t.q. \quad 0 \leq f(n) \leq c \cdot g(n) \quad \forall n \geq n_0 \}$$

Es decir, explicar qué onda con esto. Acota por arriba.

Ejemplo:

$15n^2 + 45n = O(n^2)$

$15n^2 + 45n \leq cn^2$

$15 + 45/n \leq c$

$45/n \leq c - 15$

$45/(c - 15) \leq n$// si $c > 15$, la desigualdad se queda igual

$45 \leq n$ // con $c = 16$, $n_0 = 45$

Lo que lo domina es la n^2, la otra 'n' nos da igual. Al final no es que nos importe tanto el valor final de la demostración, el punto es que encontremos un valor para demostrar que la función es n^2.

Ahora vamos con esta.

$$\Omega(g(n)) = \{ f(n): \ \exists\, c, n_0 > 0 \quad t.q. \quad 0 \leq c \cdot g(n) \leq f(n) \quad \forall\, n \geq n_0 \}$$

Es decir, explicar esto. Acota por abajo.

Ejemplo:

$15n^2 + 45n = \Omega(n^2)$

$15n^2 + 45n \geq c'n^2$

$15 + 45/n \geq c'$

$45/n \geq c' - 15$

$45/(c' - 15) \leq n$ // si $c' < 15$, la desigualdad se voltea

$45 \leq n$ // con $c' = 14$, n_0 sería - 45, pero nos quedamos con $n_0 = 0$

$$\theta(g(n)) = \{ f(n): \ \exists\, c1, c2, n_0 > 0 \quad t.q. \quad 0 \leq c1 \cdot g(n) \leq f(n) \leq c2 \cdot g(n) \quad \forall\, n \geq n_0 \}$$

Es decir, explicar esto. Acota tanto por arriba como por abajo.

Tipos de análisis.

¿El mismo algoritmo se tarda? Igual no solo importa el tamaño de entrada, sino también de la 'calidad' del mismo. Imaginemos un programa que ordena una cantidad de arreglos. Entonces también hay que considerar el tipo de dato que nos están dando, porque puede ser que en el mejor de los casos sea lineal. Pero 'el mejor de los casos' no nos da tanta información en general, pues siguiendo el ejemplo anterior, podríamos tener un arreglo completamente ordenado. El peor de los casos es más informativo, nos da

una especie de “garantía”, pues te dice algo como “no importa cómo nos des el dato, en el peor de los casos el programa es lineal”, suena mejor, ¿cierto? Entonces tenemos los siguientes tipos de análisis;

- el mejor de los casos; [nunca los vamos a ver]
- el peor de los casos;
- el caso promedio;
+ caso amortizado.

Sobre el caso amortizado. Hemos visto que un arreglo es algo maravilloso pues ¿cuánto nos tardamos en acceder a una posición? Es un tiempo constante, es muy bello. El problema con un arreglo es un “insertar”. Pero recordemos que cuando se llena el arreglo, tenemos que hacerlo el doble. Esa operación de “expande” requiere que para insertar, copie todos los elementos que ya tenía en un arreglo en uno nuevo. El inserta en un arreglo se tarda en un comportamiento lineal, pero el copiado llega a generar conflicto.

En la complejidad queremos poder decir la dificultad de un problema, no tanto de un algoritmo. Nos gustaría decir qué tan fácil es sumar o algo parecido. Un problema es tan difícil como el mejor algoritmo se tarda en completarlo. Podemos decir que dados dos números, el problema es constante debido a que tenemos un programa que cuando lo resuelve, es de manera constante. Entonces para determinar la complejidad de un problema es necesario encontrar un algoritmo que lo resuelva. Tenemos varias clases dependiendo de la complejidad del problema.

Una de ellas es la clase p. Son los problemas para los cuales existe un algoritmo de tiempo polinomial. Ya sean cuadrados, cúbicos, etcétera. Aquí vive la suma, la multiplicación de matrices. Hay otros que son de clase exp, otros de tiempo polinomial pero con una computadora cuántica, otros que son probabilísticos, etcétera.

Tenemos una clase np, podríamos pensar que son “no polinomiales”, pero es incorrecto y es penoso decirlo, así que cuidado. Lo que significa es que son del tipo de problemas que se pueden resolver por una máquina no determinista pero en tiempo polinomial. La manera más simple de verlo es que son problemas los cuales podemos verificar su solución en tiempo polinomial. Tenemos una entrada y una salida y podemos verificar que la salida es la correcta para la entrada.

La pregunta del millón es ¿**p = np**?.