Introducción al Análisis de Algoritmos

Estructuras de Datos y Algoritmos – IIC2133

¿Qué es un algoritmo?

Un conjunto de instrucciones simples claramente especificadas —o procedimiento computacional— que deben ser obedecidas para resolver un problema:

- toma un valor, o un conjunto de valores, como input, y produce un valor, o un conjunto de valores, como output o resultado

P.ej.,

- ordenar una secuencia de números en orden no decreciente
- determinar si un string es un substring de otro string

Una vez que tenemos un algoritmo para un problema y hemos establecido (de alguna manera) que es correcto, debemos determinar cuántos recursos, tales como tiempo o espacio, va a requerir:

- un algoritmo que resuelve un problema pero toma un año o necesita cientos de gigabytes de memoria no es muy útil

Definiciones de O(), Ω() y Θ()

T(N) = O(f(N)) si existen constantes positivas c y n_0 tales que T(N) ≤ cf(N) cuando $N \geq n_0$

T(N) = Ω(g(N)) si existen constantes positivas c y n_0 tales que T(N) ≥ cf(N) cuando $N \geq n_0$

T(N) = Θ(h(N)) si y sólo si T(N) = O(h(N)) y T(N) = Ω(h(N))

Estas definiciones permiten establecer un orden relativo entre funciones

Dadas dos funciones, en general no tiene sentido decir, p.ej, f(N) < g(N), pero sí podemos comparar sus tasas (relativas) de crecimiento:

- p.ej., ¿cómo comparamos 1,000N con N^2?

Funciones típicas en esta asignatura

c	constante
$\log N$	logarítmica
N	lineal
$N \log N$	
N^2	cuadrática
N^3	cúbica
2^N	exponencial

Regla 1

Si $T_1(N) = O(f(N))$ y $T_2(N) = O(g(N))$, entonces

a) $T_1(N) + T_2(N) = O(f(N) + g(N))$

b) $T_1(N)*T_2(N) = O(f(N)*g(N))$

Regla 2

Si $T(N)$ es un polinomio de grado k, entonces $T(N) = O(N^k)$

Regla 3

$\log^k N = O(N^a)$ para cualquier constante k y constante a apropiada

Es muy mal estilo incluir constantes o términos de orden inferior dentro de una O: no escribimos $T(N) = O(2N^2)$ ni $T(N) = O(N^2+N)$; en ambos casos, la forma correcta es $T(N) = O(N^2)$

Modelo de computación

Un computador normal, en que las instrucciones son ejecutadas (llevadas a cabo) secuencialmente

Instrucciones simples: suma, multiplicación, comparación, asignación

Toma exactamente una unidad de tiempo hacer cualquier cosa simple

Los números enteros son de tamaño fijo y no hay operaciones sofisticadas, tales como inversión de matrices u ordenación, que claramente no pueden ser llevadas a cabo en una unidad de tiempo

Infinita memoria

¿Qué analizamos?

El recurso más importante es generalmente el tiempo de ejecución:

- … se ve afectado por el compilador y el computador usados, con respecto a los cuales no podemos hacer nada
- … y también por el algoritmo usado y por los datos de entrada (*input*)

El tamaño N del input es la consideración más importante

Definimos las funciones $T_{avg}(N)$ y $T_{worst}(N)$, como los tiempos de ejecución que toma un algoritmo en el caso promedio y en el peor caso, respectivamente; claramente, $T_{avg}(N) \leq T_{worst}(N)$

Aunque $T_{avg}(N)$ a menudo refleja el comportamiento típico, normalmente calculamos $T_{worst}(N)$:

- $T_{worst}(N)$ proporciona una cota para cualquier input
- $T_{avg}(N)$ es normalmente mucho más difícil de calcular

Tiempos de ejecución (en segundos) de varios algoritmos para el mismo problema*

N	$O(N^3)$	$O(N^2)$	$O(N\log N)$	$O(N)$
100	0.00016	0.000006	0.000005	0.000002
1,000	0.096	0.00037	0.00006	0.00002
10,000	86.67	0.0333	0.00062	0.00022
100,000	NA	3.33	0.0067	0.0022
1,000,000	NA	NA	0.075	0.023

* Problema de la *subsecuencia de suma máxima* (ver diap. #10); los tiempos no incluyen el tiempo necesario para leer el input

Reglas para el cálculo de tiempos de ejecución

Ciclos `for`, `while`:

- A lo más es el tiempo de ejecución de las instrucciones dentro del ciclo (incluyendo los tests) multiplicado por el número de iteraciones

Ciclos anidados:

- Se analizan de adentro hacia afuera; el tiempo de ejecución de una instrucción dentro de un grupo de ciclos anidados es el tiempo de ejecución de la instrucción multiplicado por el producto de los tamaños de todos los ciclos

Instrucciones consecutivas:

- Simplemente se suman, por lo que el que cuenta es el máximo de los tiempos de ejecución de cada una de las instrucciones (ver Regla 1a, diap.#5)

`if (`*`condición`*`)` *`S1`* `else` *`S2`*:

- No más que el tiempo de ejecución del test más el tiempo de ejecución más grande de *S1* y *S2*

El problema de la subsecuencia de suma máxima

Dados los enteros (posiblemente negativos) A_1, A_2, ..., A_N,

… encontrar el valor máximo de $\sum_{k=1}^{j} A_k$

Por conveniencia, la suma es 0 si todos los enteros son negativos

P.ej., para el input –2, 11, –4, 13, –5, –2

… la respuesta es 20 (A_2 hasta A_4)