CAPITULO 3: CARACTERIZACIÓN DE LOS TIEMPOS DE EJECUCIÓN

El orden de crecimiento del tiempo de ejecución de un algoritmo, definido en el Capítulo 2, da una forma sencilla de caracterizar la eficiencia del algoritmo y también nos permite compararlo con algoritmos alternativos. Una vez que el tamaño de entrada n se vuelve lo suficientemente grande, el orden de n log n del tiempo de ejecución en el peor caso de la ordenación por mezcla supera al de la ordenación por inserción, cuyo tiempo de ejecución en el peor caso es de $O(n^2)$. Aunque a veces podemos determinar el tiempo de ejecución exacto de un algoritmo, como hicimos con la ordenación por inserción en el Capítulo 2, esa precisión rara vez vale el esfuerzo de calcularla. Para entradas lo suficientemente grandes, las constantes multiplicativas y los términos de orden inferior de un tiempo de ejecución exacto quedan dominados por los efectos del tamaño de la entrada en sí.

Cuando miramos tamaños de entrada lo suficientemente grandes como para que sea relevante solo el orden de crecimiento del tiempo de ejecución, estamos estudiando la eficiencia asintótica de los algoritmos. Esto es, nos preocupamos cómo el tiempo de ejecución de un algoritmo aumenta con el tamaño de la entrada en el límite, a medida que el tamaño de la entrada crece sin límite. Usualmente, un algoritmo que es asintóticamente más eficiente es la mejor elección para casi todos los casos excepto para entradas muy pequeñas.

Este capítulo presenta varios métodos estándar para simplificar el análisis asintótico de algoritmos. La siguiente sección presenta de manera informal los tres tipos de notación asintótica más comúnmente usados, de los cuales ya hemos visto un ejemplo en la notación O. También muestra una forma de usar estas notaciones asintóticas para razonar sobre el tiempo de ejecución en el peor caso de la ordenación por inserción. Luego examinamos las notaciones asintóticas de manera más formal y presentamos varias convenciones notacionales utilizadas a lo largo de este libro. La última sección revisa el comportamiento de funciones que comúnmente surgen al analizar algoritmos.

3.1 O-notación, Ω-notación, y Θ-notación

O-notation

La notación O caracteriza un límite superior en el comportamiento asintótico de una función. En otras palabras, indica que una función crece a una velocidad no mayor que cierta tasa, basada en el término de mayor orden. Considera, por ejemplo, la función $7n^3 + 100n^2 - 20n + 6$. Su término de mayor orden es $7n^3$, por lo que decimos que la tasa de crecimiento de esta función es n^3. Dado que esta función crece a una velocidad no mayor que n^3, podemos escribir que es $O(n^3)$. Puede sorprenderte que también podamos decir que la función $7n^3 + 100n^2 - 20n + 6$ es $O(n^4)$. ¿Por qué? Porque la función crece más lentamente que n^4, estamos correctos al decir que crece a una velocidad no mayor. Como podrías haber adivinado, esta función también es $O(n^5)$, $O(n^6)$, y así sucesivamente. Más generalmente, es O(n^c) para cualquier constante $c \geq 3$.

Notación Ω

La notación Ω caracteriza un límite inferior en el comportamiento asintótico de una función. En otras palabras, indica que una función crece al menos tan rápido como cierta tasa, basada —como en la notación O— en el término de mayor orden. Dado que la función $7n^3 + 100n^2 - 20n + 6$ crece al menos tan rápido como n^3, esta función es $\Omega(n^3)$. Esta función también es $\Omega(n^2)$ y $\Omega(n)$. Más generalmente, es $\Omega(n^c)$ para cualquier constante $c \leq 3$.

Notación Θ

La notación Θ caracteriza un límite ajustado en el comportamiento asintótico de una función. Indica que una función crece precisamente a una cierta tasa, basada —una vez más— en el término de mayor orden. Otra forma de verlo es que la notación Θ caracteriza la tasa de crecimiento de la función dentro de un factor constante por encima y dentro de un factor constante por debajo. Estos dos factores constantes no necesitan ser iguales.

Si puedes demostrar que una función es tanto O(f(n)) como Ω(f(n)) para alguna función f(n), entonces has demostrado que la función es Θ(f(n)). (La próxima sección establece este hecho como un teorema). Por ejemplo, dado que la función $7n^3 + 100n^2 - 20n + 6$ es tanto $O(n^3)$ como $\Omega(n^3)$, también es $\Theta(n^3)$.

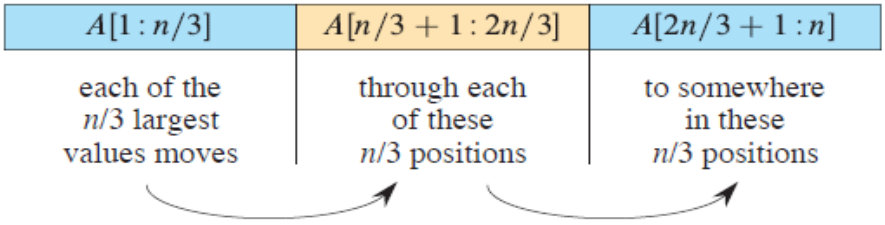
Ejemplo: Ordenación por inserción (Insertion sort)

Revisemos la ordenación por inserción y veamos cómo trabajar con la notación asintótica para caracterizar su tiempo de ejecución en el peor caso $O(n^2)$ sin evaluar sumatorias como lo hicimos en el Capítulo 2. Aquí está el procedimiento INSERTION-SORT una vez más:

```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1].
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

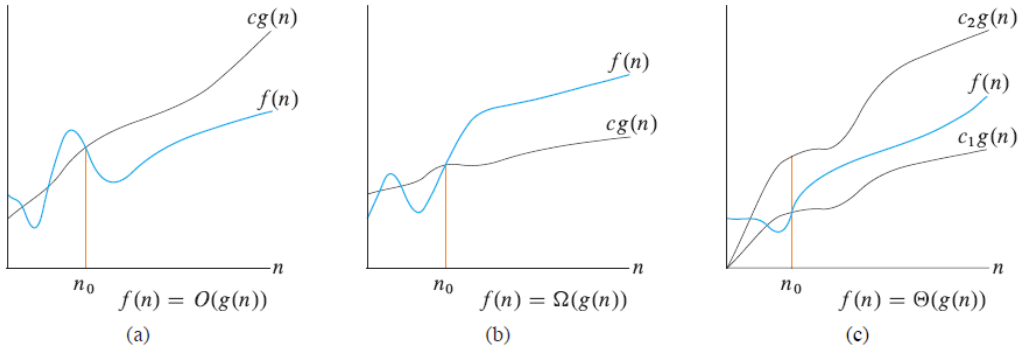
¿Qué podemos observar sobre cómo opera el pseudocódigo? El procedimiento tiene bucles anidados. El bucle exterior es un bucle for que se ejecuta n - 1 veces, independientemente de los valores que se están ordenando. El bucle interior es un bucle while, pero el número de iteraciones que realiza depende de los valores que se están ordenando. La variable de bucle j comienza en i - 1.

Y disminuye en 1 en cada iteración hasta que alcanza 0 o A[j] ≤ key. Para un valor dado de i, el bucle while podría iterar 0 veces, i - 1 veces, o cualquier valor intermedio. El cuerpo del bucle while (líneas 6-7) toma un tiempo constante por iteración del bucle while.



3.2 Notación asintótica: definiciones formales

Habiendo visto la notación asintótica de manera informal, hagámoslo más formal. Las notaciones que usamos para describir el tiempo de ejecución asintótico de un algoritmo se definen en términos de funciones cuyos dominios son típicamente el conjunto $\mathbb{N}$ de números naturales o el conjunto $\mathbb{R}$ de números reales. Dichas notaciones son convenientes para describir una función de tiempo de ejecución T(n). Esta sección define las notaciones asintóticas básicas y también introduce algunos abusos notacionales "apropiados" comunes.



Notación O

Como vimos en la Sección 3.1, la notación O describe un límite superior asintótico. Usamos la notación O para dar un límite superior en una función, dentro de un factor constante. Aquí está la definición formal de la notación O. Para una función dada g(n), denotamos por O(g(n)) (pronunciado "big-oh de g de n" o a veces simplemente "oh de g de n") el conjunto de funciones O(g(n)) = {f(n): existen constantes positivas c y n_0 tales que $0 \le f(n) \le cg(n)$ para todo $n \ge n_0$}.

Una función f(n) pertenece al conjunto O(g(n)) si existe una constante positiva c tal que $f(n) \le cg(n)$ para un n suficientemente grande. La Figura 3.2(a) muestra la intuición detrás de la notación O. Para todos los valores de n a la derecha de n_0, el valor de la función f(n) está por debajo o igual a cg(n).

La definición de O(g(n)) requiere que toda función f(n) en el conjunto O(g(n)) sea asintóticamente no negativa: f(n) debe ser no negativa cuando n es suficientemente grande. (Una función asintóticamente positiva es aquella que es positiva para todo n suficientemente grande). En consecuencia, la función g(n) misma debe ser asintóticamente no negativa, o de lo contrario el conjunto O(g(n)) estaría vacío. Por lo tanto, asumimos que toda función usada con la notación O es asintóticamente no

negativa. Esta suposición también se aplica a las otras notaciones asintóticas definidas en este capítulo.

Podrías sorprenderte de que definamos la notación O en términos de conjuntos. De hecho, podrías esperar que escribiríamos "$f(n) \in O(g(n))$" para indicar que f(n) pertenece al conjunto O(g(n)). En cambio, usualmente escribimos "$f(n) = O(g(n))$" y decimos "f(n) es big-oh de g(n)" para expresar la misma noción. Aunque puede parecer confuso al principio abusar de la igualdad de esta manera, veremos más adelante en esta sección que hacerlo tiene sus ventajas.

Notación Ω

Así como la notación O proporciona un límite superior asintótico en una función, la notación Ω proporciona un límite inferior asintótico. Para una función dada g(n), denotamos por Ω(g(n)) (pronunciada "big-omega de g de n" o a veces simplemente "omega de g de n") el conjunto de funciones Ω(g(n)) = {f(n): existen constantes positivas c y n_0 tales que $0 \le cg(n) \le f(n)$ para todo $n \ge n_0$}.

La Figura 3.2(b) muestra la intuición detrás de la notación Ω. Para todos los valores de n a la derecha de n_0, el valor de f(n) está por encima o igual a cg(n).

Ya hemos demostrado que $4n^2 + 100n + 500 = O(n^2)$. Ahora demostremos que $4n^2 + 100n + 500 = \Omega(n^2)$. Necesitamos encontrar constantes positivas c y n_0 tales que $4n^2 + 100n + 500 \ge cn^2$ para todo $n \ge n_0$. Como antes, dividimos ambos lados por n^2, dando $4 + 100/n + 500/n^2 \ge c$.

Esta desigualdad se cumple cuando n_0 es cualquier entero positivo y c = 4.

¿Qué pasa si hubiéramos restado los términos de menor orden del término $4n^2$ en lugar de sumarlos? ¿Qué tal si tuviéramos un coeficiente pequeño para el término n^2?

La función $n^2/100 - 100n - 500 = \Omega(n^2)$. Dividiendo por n^2 da $1/100 - 100/n - 500/n^2 \ge c$. Podemos elegir un valor para n_0 que sea al menos 10,005 y encontrar un valor positivo para c. Por ejemplo, cuando $n_0 = 10{,}005$, podemos elegir $c = 2.49 \times 10^{-9}$. Sí, ese es un valor muy pequeño para c, pero es positivo. Si seleccionamos un valor mayor para n_0, también podemos aumentar c. Por ejemplo, si $n_0 = 100{,}000$, entonces podemos elegir c = 0.0089. Cuanto mayor sea el valor de n_0, más cerca estaremos del coeficiente 1/100 y podremos elegir c.

Notación Θ

Usamos la notación Θ para límites asintóticos ajustados. Para una función dada g(n), denotamos por Θ(g(n)) ("theta de g de n") el conjunto de funciones

Θ(g(n)) = {f(n): existen constantes positivas c_1, c_2, y n_0 tales que $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ para todo $n \ge n_0$}.

La Figura 3.2(c) muestra la intuición detrás de la notación Θ. Para todos los valores de n a la derecha de n_0, el valor de f(n) está en o por encima de $c_1 g(n)$ y en o por debajo de $c_2 g(n)$. En otras palabras, para todo $n \ge n_0$, la función f(n) es igual a g(n) dentro de factores constantes.

Las definiciones de las notaciones O, Ω, y Θ llevan al siguiente teorema, cuya demostración dejamos como Ejercicio 3.2-4.

TEOREMA 3.1

Para cualesquiera dos funciones f(n) y g(n), tenemos f(n) = Θ(g(n)) si y solo si f(n) = O(g(n)) y f(n) = Ω(g(n)).

Típicamente aplicamos el Teorema 3.1 para probar límites asintóticos ajustados a partir de límites superiores e inferiores asintóticos.

Notación asintótica y tiempos de ejecución

Cuando uses la notación asintótica para caracterizar el tiempo de ejecución de un algoritmo, asegúrate de que la notación asintótica que uses sea lo más precisa posible sin exagerar a qué tiempo de ejecución se aplica. Aquí hay algunos ejemplos de uso correcto e incorrecto de la notación asintótica para caracterizar tiempos de ejecución.

Comencemos con la ordenación por inserción. Podemos decir correctamente que el tiempo de ejecución en el peor caso de la ordenación por inserción es $O(n^2)$, $\Omega(n^2)$, y —por el Teorema 3.1— $\Theta(n^2)$. Aunque las tres formas de caracterizar los tiempos de ejecución en el peor caso son correctas, el límite $\Theta(n^2)$ es el más preciso y, por lo tanto, el más preferido. También podemos decir correctamente que el tiempo de ejecución en el mejor caso de la ordenación por inserción es $O(n)$, $\Omega(n)$, y $\Theta(n)$, siendo $\Theta(n)$ nuevamente el más preciso y, por lo tanto, el más preferido.

Aquí está lo que no podemos decir correctamente: el tiempo de ejecución de la ordenación por inserción es $\Theta(n^2)$. Eso es un error porque, al omitir "peor caso" de la afirmación, dejamos un espacio en blanco que cubre todos los casos. El error aquí es que la ordenación por inserción no se ejecuta en tiempo $\Theta(n^2)$ en todos los casos, ya que, como hemos visto, se ejecuta en tiempo $\Theta(n)$ en el mejor caso. Podemos decir correctamente que el tiempo de ejecución de la ordenación por inserción es $O(n^2)$, sin embargo, porque en todos los casos, su tiempo de ejecución no crece más rápido que n^2. Cuando decimos $O(n^2)$ en lugar de $\Theta(n^2)$, no hay problema en tener casos en los que el tiempo de ejecución crece más lentamente que n^2. Asimismo, no podemos decir correctamente que el tiempo de ejecución de la ordenación por inserción es $\Theta(n)$, pero podemos decir que su tiempo de ejecución es $\Omega(n)$.

¿Qué pasa con la ordenación por mezcla?

Dado que la ordenación por mezcla se ejecuta en tiempo $\Theta(n \lg n)$ en todos los casos, podemos simplemente decir que su tiempo de ejecución es $\Theta(n \lg n)$ sin especificar peor caso, mejor caso, o cualquier otro caso.

Notación o

El límite superior asintótico proporcionado por la notación O puede o no ser asintóticamente ajustado. El límite $2n^2 = O(n^2)$ es asintóticamente ajustado, pero el límite $2n = O(n^2)$ no lo es. Usamos la notación o para denotar un límite superior que no es asintóticamente ajustado. Formalmente definimos o(g(n)) ("little-oh de g de n") como el conjunto

o(g(n)) = {f(n): para cualquier constante c > 0, existe una constante n_0 > 0 tal que 0 ≤ f(n) < cg(n) para todo n ≥ n_0}.

Por ejemplo, 2n = o(n²), pero 2n² ≠ o(n²).

Las definiciones de la notación O y la notación o son similares. La principal diferencia es que en f(n) = O(g(n)), el límite 0 ≤ f(n) ≤ cg(n) se cumple para alguna constante c > 0, pero en f(n) = o(g(n)), el límite 0 ≤ f(n) < cg(n) se cumple para todas las constantes c > 0. Intuitivamente, en la notación o, la función f(n) se vuelve insignificante relativa a g(n) a medida que n crece:

límite (n→∞) f(n) / g(n) = 0.

Algunos autores usan este límite como definición de la notación o, pero la definición en este libro también restringe las funciones anónimas a ser asintóticamente no negativas.

Notación ω

Por analogía, la notación ω es a la notación Ω como la notación o es a la notación O. Usamos la notación ω para denotar un límite inferior que no es asintóticamente ajustado. Una forma de definirlo es

f(n) ∈ ω(g(n)) si y solo si g(n) ∈ o(f(n)).

Formalmente, sin embargo, definimos ω(g(n)) ("little-omega de g de n") como el conjunto

ω(g(n)) = {f(n): para cualquier constante c > 0, existe una constante n_0 > 0 tal que 0 ≤ cg(n) < f(n) para todo n ≥ n_0}.

Mientras que la definición de la notación o indica que f(n) < cg(n), la definición de la notación ω indica lo opuesto: que cg(n) < f(n). Por ejemplo, n²/2 = ω(n), pero n²/2 ≠ ω(n²). La relación f(n) = ω(g(n)) implica que

lím (n→∞) f(n) / g(n) = ∞,

si el límite existe. Esto es, f(n) se vuelve arbitrariamente grande en relación con g(n) a medida que n crece.

CAPÍTULO 3: NOTACIÓN ASINTÓTICA

3.1. Introducción

Un aspecto importante de este libro es el que concierne a la determinación de la eficiencia de algoritmos. En la Sección 2.3, veíamos que este conocimiento puede ayudarnos, por ejemplo, a elegir uno entre varios algoritmos en pugna. Recordemos que deseamos determinar matemáticamente la cantidad de recursos que necesita el algoritmo como función del tamaño (o a veces del valor) de los casos considerados. Dado que no existe una computadora estándar con la cual se puedan comparar todas las medidas del tiempo de ejecución, vimos también en la Sección 2.3 que nos contentaremos con expresar el tiempo requerido por el algoritmo salvo por una constante multiplicativa.

Con este objetivo, presentaremos ahora formalmente la notación asintótica que se utiliza a lo largo de todo el libro. Además, esta notación permite realizar simplificaciones sustanciales aun cuando estemos interesados en medir algo más tangible que el tiempo de ejecución, tal como el número de veces que se ejecuta una instrucción dada dentro de un programa.

Esta notación se denomina «asintótica» porque trata acerca del comportamiento de funciones en el límite, esto es, para valores suficientemente grandes de su parámetro. En consecuencia, los argumentos basados en la notación asintótica pueden no llegar a tener un valor práctico cuando el parámetro adopta valores «de la vida real». Sin embargo, las enseñanzas de la notación asintótica suelen tener una relevancia significativa. Esto se debe a que, como regla del pulgar, un algoritmo que sea superior asintóticamente suele ser (aunque no siempre) preferible incluso en casos de tamaño moderado.

Diversas notaciones asintóticas:

- Notación para “el orden de”
- Notación omega
- Notación theta
- **Notación asintótica condicional**
  Hay muchos algoritmos que resultan más fáciles de analizar si en un principio limitamos nuestra atención a aquellos casos cuyo tamaño satisfaga una cierta condición, tal como ser una potencia de 2.
  Considérese por ejemplo el algoritmo **divide y vencerás** para multiplicar enteros grandes. Sea n el tamaño de los enteros que hay que multiplicar (supongamos que eran de un mismo tamaño). El algoritmo se ejecuta directamente si n=1n = 1n=1, lo cual requiere a microsegundos para una constante apropiada a. Si n>1, el algoritmo se ejecuta recursivamente multiplicando cuatro parejas de enteros de tamaño [n/2].
  Además, es precisa una cantidad lineal de tiempo para efectuar tareas adicionales. Por sencillez, digamos que el trabajo adicional requiere de n microsegundos para una constante adecuada b (para ser exactos, sería necesario un tiempo entre $b_1 n$ y $b_2 n$ microsegundos para las constantes adecuadas b_1 y b_2.

- **Notación asintótica con varios parámetros**
  Puede suceder, cuando se analiza un algoritmo, que su tiempo de ejecución dependa simultáneamente de más de un parámetro del ejemplar en cuestión. Esta situación es típica de ciertos algoritmos para problemas de grafos, por ejemplo, en los cuales el tiempo depende tanto del número de nodos como del número de aristas. En tales casos, la noción de «tamaño del ejemplar» que se ha utilizado hasta el momento puede perder gran parte de su significado. Por esta razón, se generaliza la notación asintótica de forma natural para funciones de varias variables.
- **Operaciones sobre notación asintótica**
  Para simplificar algunos cálculos, podemos manipular la notación asintótica usando operadores aritméticos. Por ejemplo, $O(f(n))+O(g(n))$ representa el conjunto de operaciones obtenidas al sumar, punto por punto, cualquier función de $O(f(n))$ con cualquier función de $O(g(n))$.
  Intuitivamente, este conjunto representa el orden del tiempo requerido por un algoritmo compuesto por dos fases:

1. Una primera fase con tiempo del orden de $f(n)$.
2. Una segunda fase con tiempo del orden de $g(n)$.

  Aunque las constantes ocultas que multiplican a $f(n)$ y $g(n)$ pueden ser diferentes, esto no afecta el resultado, ya que se puede demostrar que:
  $O(f(n))+O(g(n))$ es idéntico a $O(f(n) + g(n))$.
  Además, por la regla del máximo, esto equivale a: $O(\max(f(n),g(n)))$, o, si se prefiere: $\max(O(f(n)),O(g(n)))$.
  En resumen, estas simplificaciones permiten analizar algoritmos de manera más eficiente sin perder rigor matemático.