Análisis Básico de Algoritmos

Notación de Landau

La notación de Landau "captura" el término dominante en una función, es decir, el que más influye en el valor de la función.

- Cota superior: $O(f) = \{ g / g \text{ no crece más que } f \}$
- Cota inferior: $\Omega(f) = \{ g / g \text{ crece al menos como } f \}$
- Cota exacta: $\Theta(f) = \{ g / g \text{ crece exactamente como } f \}$

Órdenes de Complejidad

Con la medida O (cota superior) se pueden clasificar las funciones en familias según se pueda acotar su crecimiento.

Jerarquía de Órdenes de Complejidad

$$O(1) \subset O(\log_n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \cdot \log_n) \subset O(n^2) \subset$$
$$\subset O(n^3) \subset \cdots \subset O(n^k) \subset O(2^n) \subset \cdots \subset O(k^n) \subset O(n!) \subset O(n^n)$$

Reglas Prácticas para el Cálculo del Coste

El primer paso para realizar un correcto cálculo del coste es Identificar el Tamaño del Problema. Sin esta identificación, cualquier cálculo posterior sería incorrecto.

Algoritmos Iterativos

En los algoritmos iterativos las diferentes sentencias se ejecutan en la secuencia en la que se han escrito.

Operaciones básicas

No dependen del tamaño del problema. Son:

- Operaciones de Entrada/Salida
- Asignaciones
- Cálculo de expresiones que no dependan del tamaño del problema

Tienen un coste constante con respecto al tamaño del problema.

Composición secuencial de sentencias

Sumar el coste individual de cada una de ellas. Es decir, escoger el coste más grande de todas ellas.

Sentencias condicionales

- **if** (e) {S_1} **else** {S_2} → $\max\{O(C_e), O(C_{S1}), O(C_{S2})\}$
- **switch** (e){
  case v_1 : S_1;
  case v_2 : S_2;
  …
  case v_n : S_n;
  } → $\max\{O(C_e), O(C_{S1}), O(C_{S2}), \ldots, O(C_{Sn})\}$

Como regla general, escogemos el coste mayor de la evaluación de la expresión y todas las posibles secuencias de sentencias que se ejecuten en la instrucción condicional.

Bucles

Considerando que el número de vueltas viene dado por la función v(n):

- **for** (ini;e;inc) {S} → $\max\{\{O(c_{ini}),O(v(n))\cdot\max\{O(c_e),O(c_S),O(c_{inc})\}\}$
- **while** (e) {S} → $O(v(n)\cdot\max\{O(c_e),O(c_S)\}$
- **do** {S} **while** (e) → $O(v(n)\cdot\max\{O(c_e),O(c_S)\}$

Como regla general, multiplicamos el número de vueltas por el coste de cada vuelta y sumamos (en caso de que esté incluida en el bucle) el coste de la inicialización.

Llamadas a subprogramas

- $f(e_1,e_2,\ \dots\ ,e_n)$ → $\max\{O(c_{e1}),O(c_{e2}),\dots,O(c_{en}),O(c_f)\}$

Podemos verlo como la composición secuencial de la evaluación de cada parámetro y, a continuación, la ejecución del cuerpo del subprograma.

Algoritmos Recursivos

Los algoritmos recursivos se definen en función de sí mismos:

- Existe una llamada recursiva que resuelve un subproblema (o varios) más pequeño que el problema actual.
- Los resultados devueltos por la recursión son combinados para producir la solución del problema actual.

La función de coste también es recursiva y se denomina recurrencia. Resolver de forma exacta una recurrencia puede ser muy complejo, así que simplemente calcularemos a qué orden de complejidad pertenece.

Reducción mediante sustracción

$$T(n)=\begin{cases} c_b(n) & si\ 0\le n<b \\ a\cdot T(n-b)+c_{nr}(n) & si\ n\ge b \end{cases} \qquad T(n)\in\begin{cases} O(n\cdot c_{nr}(n)+c_b(n)) & si\ a=1 \\ O(a^{(n\ \mathbf{div}\ b)}\cdot(c_{nr}(n)+c_b(n))) & si\ a>1 \end{cases}$$

Forma general de la recurrencia — Reglas prácticas para el cálculo del orden

- $c_b(n)$ → coste del caso base
- $c_{nr}(n)$ → coste de las operaciones no recursivas en cada llamada
- a → número de llamadas recursivas
- n-b → tamaño de los subproblemas

Reducción mediante división

$$T(n)=\begin{cases} c_b(n) & si\ 1\le n<b \\ a\cdot T(n\ \mathbf{div}\ b)+c_{nr}(n) & si\ n\ge b \end{cases} \qquad T(n)\in\begin{cases} O(\log_b(n)\cdot c_{nr}(n)+c_b(n)) & si\ a=1 \\ O(n^{\log_b(a)}\cdot(c_{nr}(n)+c_b(n))) & si\ a>1 \end{cases}$$

Forma general de la recurrencia — Reglas prácticas para el cálculo del orden

- $c_b(n)$ → coste del caso base
- $c_{nr}(n)$ → coste de las operaciones no recursivas en cada llamada
- a → número de llamadas recursivas
- n **div** b → tamaño de los subproblemas

Algunos casos prácticos habituales de cálculo del coste de algoritmos recursivos

- Reducción mediante sustracción
  - Recursión lineal (a = 1)
    - $c_b(n) \in O(1)$ → $T(n) \in O(n \cdot c_{nr}(n))$
    - $c_{nr}(n) \in O(n^k)$ y $c_b(n) \in O(1)$ → $T(n) \in O(n^{k+1})$
  - Recursión múltiple (a > 1)
    - $c_b(n) \in O(1)$ → $T(n) \in O(a^{n \text{ div } b} \cdot c_{nr}(n))$
    - $b = 1$ y $c_b(n), c_{nr}(n) \in O(1)$ → $T(n) \in O(a^n)$
- Reducción mediante división
  - Recursión lineal (a = 1)
    - $c_b(n) \in O(1)$ → $T(n) \in O(\log_b(n) \cdot c_{nr}(n))$
    - $c_{nr}(n) \in O(n^k)$ y $c_b(n) \in O(1)$ → $T(n) \in O(n^k \cdot \log_b(n))$
  - Recursión múltiple (a > 1)
    - $c_b(n) \in O(1)$ → $T(n) \in O(a^{n \text{ div } b} \cdot c_{nr}(n))$
    - $a = b = 2$ y $c_b(n), c_{nr}(n) \in O(1)$ → $T(n) \in O(n^{\log_2(2)}) = O(n)$