Universidad Diego Portales

Apunte Clase 3: Análisis de Algoritmos

Profesor Yerko Ortiz
2024

Contents

1 Análisis de algoritmos

El análisis de algoritmos es la rama de la computación en la que dado un algoritmo, busca determinar sus características de rendimiento respecto al uso de recursos como CPU o memoria.

2 Benchmark

Para saber cuántos recursos gasta un algoritmo en particular es posible realizar un benchmark. El benchmark más básico que podemos realizar en Java es medir cuánto demora en ejecutar un método respecto el tamaño de entrada. Para esto se puede hacer uso del método nanoSeconds perteneciente a la clase System.
Por ejemplo para medir cuánto demora un método estático cualquiera (llamado f en este caso):

```
long startTime = System.nanoTime();
f(102032);
long endTime = System.nanoTime();
```

Para saber cuánto demoro su ejecución basta con calcular la diferencia entre endTime y startTime.
Ahora bien, es cierto que realizar benchmarks o profiling en muchos proyectos de ingeniería es una necesidad, pero los resultados tienen un sesgo importante, dependen de las especificaciones de la maquina que ejecuta las pruebas por lo que no es fiable confiar en benchmarks para el análisis de un algoritmo que ha de ser implementado en múltiples lenguajes y ejecutado en múltiples dispositivos.

3 Contar instrucciones

Para eliminar la dependencia de las especificaciones de la maquina que ejecuta un determinado algoritmo, es posible contar la cantidad de instrucciones y asociar dicho calculo a alguna función $T(N)$.

3.1 Análisis de búsqueda lineal

```
static boolean LinearSearch(int[] A, int x) {
    for(int i = 0; i < A.length; i++) { //depende del tamaño de A
        if (A[i] == x){ // 1
            return true; // 1
        }
    }
    return false; //1
}
```

En dicho ejemplo podemos decir que la cantidad de instrucciones depende del tamaño del arreglo A, si definimos una función $T(N)$ que esté en función de

dicho tamaño podemos decir que:

$$T(N) = 2N + 1, \text{ donde } N \text{ caracteriza el tamaño de A}$$

3.2 Análisis de transformar un entero a binario

```
static String intToBin(int N){
    String s = ""; \\1
    s += (N % 2); \\1
    while(N > 1) { \\ ??
        N /= 2; \\ 1
        s = (N % 2) + s; \\ 1*
    }
    return s; \\ 1
}
```

Para este análisis se asumirá que concatenar Strings en Java con el símbolo + toma tiempo constante, en la práctica dada la naturaleza inmutable de los Strings en el lenguaje, utilizar dicho operador toma una cantidad lineal de tiempo porque es necesario crear un nuevo String y copiar todos los caracteres. Todo lo que queda para analizar dicho algoritmo es encontrar una expresión que describa el número de iteraciones que habrán en función de N.
Si N = 8, entonces habrán 3 iteraciones (8, 4, 2).
Si N = 1024, entonces habrán 10 iteraciones (1024, 512, 256, 128, 64, 32, 16, 8, 4, 2).
Este es un comportamiento logarítmico, puesto que en cada iteración el tamaño de entrada disminuye a la mitad. Por lo que se puede afirmar que la incógnita de cuántas iteraciones realiza ese ciclo es en función de $\log n$.
Entonces podemos expresar que el tiempo de ejecución de transformar un entero a binario es:

$$T(N) = 2 \log N + 3$$

Ahora bien, este análisis entrega la caracterización del tiempo de ejecución del algoritmo con una función $T(N)$, pero la pregunta del millón es si realmente necesitamos las constantes aditivas o multiplicativas que acompañan a $\log n$.

4 Notación Big Oh

La notación Big Oh caracteriza una función $f(n)$ mediante una cota superior.

$$\mathcal{O}(g(n)) = \{f(n) : \text{ existen las constantes positivas } c \text{ y } n_0,$$

$$\text{tal que } 0 \leq \ f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$

La idea se entiende mejor con un ejemplo, sea $f(n) = 600n^2+20n+800$, entonces podemos afirmar que $600n^2 + 20n + 800 = \mathcal{O}(n^2)$ puesto que si escogemos una

constante multiplicativa c lo suficientemente grande $c(n^2)$ tendrá mayor crecimiento que $f(n)$ a partir de un determinado n_0.
Ahora bien, en la práctica para calcular la cota superior de una función utilizando la notación Big Oh hay que seguir los siguientes pasos:

1. Eliminar las constantes multiplicativas
2. Eliminar los términos de menor grado

Esta notación es de especial uso a la hora de analizar algoritmos puesto que denota el peor caso de un determinado algoritmo.
A modo de ilustración, los siguientes ejemplos le ayudarán a entender cómo aplicar la notación:

1. $f(n) = 200\sqrt{n} + \frac{1}{2}\log n + 8 = \mathcal{O}(\sqrt{n})$ Explicación: se escogió $\sqrt{n}$ puesto que tiene mayor crecimiento que $\log n$
2. $f(n) = 2n^3 + 12000n^2 + 90000 = \mathcal{O}(n^3)$ Explicación n^3 tiene mayor grado de crecimiento que el resto.
3. $f(n, m) = 20n^2 + 5m + 10n + 9 = \mathcal{O}(n^2 + m)$ Explicación: en este ejemplo la función depende de dos variables que describen el tamaño de la entrada, en casos como estos se debe expresar la cota superior en función de todas las variables que influyen en el crecimiento de la entrada del algoritmo.
4. $f(n) = 50 \cdot 2^n + 3^n + 500n^{24} = \mathcal{O}(3^n)$ Explicación: en este caso se escoge 3^n puesto que tiene mayor base y eso implica que crece más rápido que 2^n.

Ahora con esta idea podemos expresar el tiempo de ejecución de los algoritmos que estudiaremos de mejor forma:

- Búsqueda lineal: $T(N) = \mathcal{O}(N)$
- Transformar entero a binario: $T(N) = \mathcal{O}(\log n)$