EFICIENCIA DE UN ALGORITMO

Abraham Sánchez López
FCC/BUAP
Grupo MOVIS

MOVIS

Facultad de Ciencias de la Computación

Qué se estudia en este tema?

1. Comparación de algoritmos
2. Principio de invarianza
3. Eficiencia
4. Notaciones asintóticas
5. Cálculo de la eficiencia de un algoritmo
6. Resolución de ecuaciones en recurrencia:
7. Aplicaciones

Comparación de algoritmos, I

- Habitualmente disponemos de más de un algoritmo para resolver un problema dado, ¿Cuál elegimos?
- Uso de recursos
- Computacionales
  - Tiempo de ejecución
  - Espacio en memoria
- No computacionales
  - Esfuerzo de desarrollo (análisis, diseño e implementación)
- Factores que influyen en el uso de recursos
- Recursos computacionales
  - Diseño del algoritmo
  - Complejidad del problema (tamaño de las entradas)
  - Hardware (arquitectura, Mhz, MBs,…)
  - Calidad del código
  - Calidad del compilador (optimizaciones)

Comparación de algoritmos, II

- Recursos no computacionales
  - Complejidad del algoritmo
  - Disponibilidad de bibliotecas reutilizables

Principio de invarianza

- Dos implementaciones de un mismo algoritmo no diferirán más que en una constante multiplicativa.

- Si $t_1(n)$ y $t_2(n)$ son los tiempos de dos implementaciones de un mismo algoritmo, se puede comprobar que:

$$\exists c, d \in \mathcal{R}, t_1(n) \leq c t_2(n);\ t_2(n) \leq d t_1(n)$$

Eficiencia, I

- Medida del uso de los recursos computacionales requeridos por la ejecución de un algoritmo en función del tamaño de las entradas.
- $T(n)$ Tiempo empleado para ejecutar el algoritmo con una entrada de tamaño n

- Tipos de análisis

  ¿Cómo medimos el tiempo de ejecución de un algoritmo?

  - **Mejor caso**: En condiciones óptimas (no se usa por ser demasiado optimista).
  - **Peor caso**: En el peor escenario posible (nos permite acotar el tiempo de ejecución).
  - **Caso promedio**: Caso difícil de caracterizar en la práctica.
  - **Análisis probabilístico**: Asume una distribución de probabilidad sobre las posibles entradas.
  - **Análisis amortizado**: Tiempo medio de ejecución por operación sobre una secuencia de ejecuciones sucesivas.

Eficiencia, II

- Algunos ejemplos:

- *Algoritmo 1*:

  $T(n) = 10^{-4}\ 2^n$ segundos

  $n = 38$ datos

  $T(n) = 1$ año

- *Algoritmo 2*:

  $T(n) = 10^{-2}\ n^3$ segundos

  $n = 1000$ bits

  $T(n) = 1$ año

- ¿Cuál es mejor?
- Se precisa de un **análisis asintótico**!

Notaciones asintóticas, I

- Estudian el comportamiento del algoritmo cuando el tamaño de las entradas es lo suficientemente grande, sin tener en cuenta lo que ocurre para entradas pequeñas y obviando factores constantes.

Orden de eficiencia

- Un algoritmo tiene un tiempo de ejecución de **orden *f*(*n*)**, para una función dada *f*, si existe una constante positiva *c* y una implementación del algoritmo capaz de resolver cada caso del problema en un tiempo acotado superiormente por $c \cdot f(n)$, donde *n* es el tamaño del problema considerado.

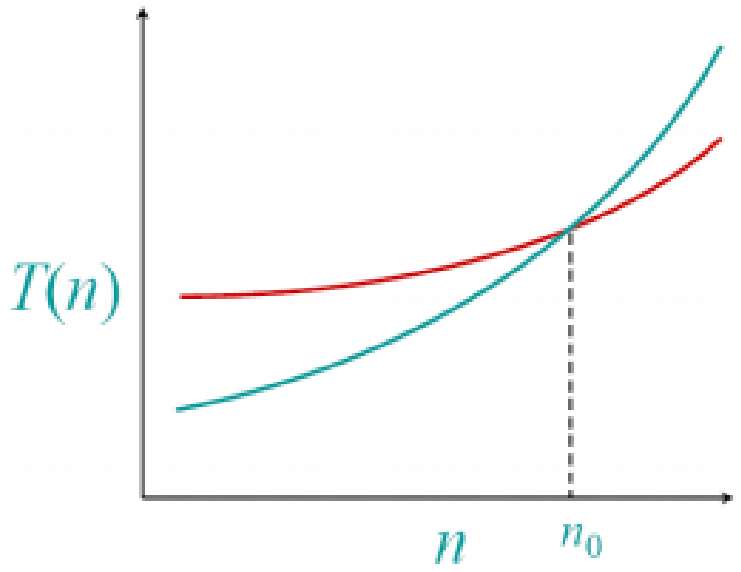
Notación *O*

- Decimos que una función $T(n)$ es $O(f(n))$ si existen constantes n_0 y c tales que $T(n) \leq c \cdot f(n)$ para todo $n \geq n_0$:

$$T(n) \text{ es } O\big(f(n)\big) \Leftrightarrow \exists c \in \Re, \exists n_0 \in \mathbb{N}, \text{tal que } \forall n > n_0 \in \mathbb{N}, T(n) \leq cf(n)$$

Notaciones asintóticas, II

- Ejemplos
- $T(n) = 3n$
  $T(n)$ es $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$ y $O(2^n)$.
- $T(n) = (n+1)^2$
  $T(n)$ es $O(n^2)$, $O(n^3)$ y $O(2^n)$.
  $T(n)$ no es $O(n)$ ni $O(n \log n)$.
- $T(n) = 32n^2 + 17n + 32$
  $T(n)$ es $O(n^2)$ pero no es $O(n)$.
- $T(n) = 3n^3 + 345n^2$
  $T(n)$ es $O(n^3)$ pero no es $O(n^2)$.
- $T(n) = 3^n$
  $T(n)$ es $O(3^n)$ pero no es $O(2^n)$.



Notaciones asintóticas, III

- Notación Ω (cota inferior)

  $T(n)$ es $\Omega(f(n))$ cuando $\exists c \in \Re^+, \exists n_0 \in \mathbb{N}: \forall n \geq n_0 \Rightarrow T(n) \geq cf(n)$

- Notación Θ (orden exacto)

  $T(n)$ es $\Theta(f(n))$ cuando $T(n)$ es $O(f(n))$ y $T(n)$ es $\Omega(f(n))$.

- Ordenes de eficiencia más comunes

N	$O(\log_2 n)$	$O(n^2)$	$O(n \log_2 n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
10	3 µs	10 µs	30 µs	0.1 ms	1 ms	4 s
25	5 µs	25 µs	0.1 ms	0.6 ms	33 s	10^{11} años
50	6 µs	50 µs	0.3 ms	2.5 ms	36 años	...
100	7 µs	100 µs	0.7 ms	10 ms	10^{17} años	...
1000	10 µs	1 ms	10 ms	1 s	...	...
10000	13 µs	10 ms	0.1 s	100 s	...	...
100000	17 µs	100 ms	1.7 s	3 horas	...	...
1000000	20 µs	1 s	20 s	12 días	...	...

- Tiempos calculados suponiendo 1 µs por operación elemental.

$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset O(2^n) \subset O(n!)$

Reglas para el cálculo del orden, I

- A continuación presentamos las reglas que nos permiten obtener el orden de complejidad de un algoritmo.
- Las instrucciones que tenemos en el lenguaje, junto con su costo, son las siguientes:
- Operaciones aritméticas elementales sobre números: suma, resta, producto y división. Consideramos que cada una de estas operaciones tiene un costo constante, por lo que todas ellas está en $\Theta(1)$.
- Comparaciones entre datos, como por ejemplo decidir si un número es mayor que otro, o si dos números son iguales. Cada test de esta forma tiene también costo constante.
- Asignaciones de la forma $x := e$. Suponiendo que las operaciones que pueden aparecer en e tienen costo constante, el costo de toda la asignación será también constante. Si no, el costo de la asignación será el costo de la evaluación de e. Lo mismo ocurre en el caso de las asignaciones múltiples.
- Composición secuencial $I_1 ; I_2$. Como el costo intenta medir una aproximación del tiempo de ejecución, para medir el tiempo total de $I_1 ; I_2$ hay que sumar los tiempos de I_1 e I_2.

Reglas para el cálculo del orden, II

- Instrucciones condicionales de la forma **si** B **entonces** I_1 **si no** I_2 **fin si**. El costo es la suma del costo de evaluar la condición booleana B (generalmente constante) más el costo de ejecutar I_1 o I_2.
- Instrucciones alternativas de la forma **casos** $B_1 \rightarrow I_1 \square \ldots \square B_n \rightarrow I_n$ **fin casos**. El costo corresponde al costo de evaluación de las guardas (generalmente constante) más el costo de ejecución de la instrucción I_j asociada a la guarda cierta.
- Ciclos de la forma **mientras** B **hacer** I **fin mientras**. Para calcular el costo de tal ciclo hay que calcular primero el costo de cada pasada y después sumar los costos de todas las pasadas que se hagan por el ciclo. El número de estas dependerá de lo que tarde en hacerse falso B, teniendo en cuenta los datos concretos sobre los que se ejecute el programa y lo grandes que sean.
- En los tres últimos tipos de instrucciones, la ejecución depende del valor de las expresiones booleanas que pueden ser cierta o falsa.
- Por lo tanto, puede ocurrir que en una ejecución el costo sea muy diferente del de la otra.

Reglas para el cálculo del orden, III

- Para proporcionar entonces una medida abstracta del costo que sea independiente de los valores concretos de datos, se consideran el costo en el peor caso y el costo en el mejor caso.
- El primero estima el tiempo máximo que habría que esperar para que acabe la ejecución del algoritmo, y el segundo, el tiempo mínimo.
- Cuando hay mucha diferencia entre ambos, es útil (aunque mucho más complejo en general) calcular el costo en el caso promedio, es decir, la suma de los costos en todos los casos (con el mismo tamaño) dividida por el número de casos.

La multiplicación de matrices, I

- Obtener *O* y Ω *para el algoritmo de la multiplicación de matrices cuadradas.*

```
for i= 1 to n
  for j=1 to n
     suma=0
     for k=1 to n
        suma=suma+a[i,k]b[k,j]
     end
     c[i,j]=suma
  end
end
```

- Podemos observar que siempre se ejecutan las mismas instrucciones, independientemente de la entrada ➔ O(*f*) = Ω (*f*) = Θ(*f*).

La multiplicación de matrices, II

- Por lo tanto:

$$t(n) = c_1 + \sum_{i=i}^{n} \left(c_2 + \sum_{j=1}^{n} \left(c_3 + \sum_{k=1}^{n} c_4 \right) \right) =$$

$$c_1 + \sum_{i=i}^{n} \left(c_2 + \sum_{j=1}^{n} \left(c_3 + c_4 n \right) \right) =$$

$$c_1 + \sum_{i=i}^{n} \left(c_2 + c_3 n + c_4 n^2 \right) = c_1 + c_2 n + c_3 n^2 + c_4 n^3$$

$$t \in \Theta(n^3)$$

Ejercicio 1, I

- El método de ordenación por Inserción realiza n–1 iteraciones sobre el vector, dejando en la i-ésima etapa ($2 \leq i \leq n$) ordenado el sub vector $a[1..i]$.
- La forma de como lo hace el algoritmo, es colocando en cada iteración el elemento $a[i]$ en su sitio correcto, aprovechando el hecho de que el sub vector $a[1..i–1]$ ya ha sido previamente ordenado.
- Este método puede ser implementado de forma iterativa como sigue:

```
PROCEDURE Insercion(VAR a:vector;prim,ult:INTEGER);
  VAR i, j, x:INTEGER;
BEGIN
    FOR i:=prim+1 TO ult DO
        x:=a[i]; j:=i-1;
        WHILE (j>=prim) AND (x<a[j]) DO
            a[j+1]:=a[j]; DEC(j)
        END;
        a[j+1]:=x
    END
END Insercion;
```

Ejercicio 1, II

- Para estudiar su complejidad, vamos a revisar los casos mejor, peor y promedio de la llamada al procedimiento Insercion(a, 1, n).
- En el mejor caso, el ciclo interno no se realiza nunca, y por lo tanto:

$$T(n) = \left(\sum_{i=2}^{n}(3+4+4+3)\right) + 3 = 14n - 11$$

- En el peor caso, hay que llevar cada elemento hasta su posición final, con lo que el ciclo interno se realiza siempre de i-1 veces. Así, en este caso se tiene:

$$T(n) = \left(\sum_{i=2}^{n}\left(3+4+\left(\sum_{j=1}^{i-1}(4+5)\right)+1+3\right)\right) + 3 = \frac{9}{2}n^2 + \frac{13}{2}n - 10$$

- En el caso promedio, podríamos suponer equiprobable la posición de casa elemento dentro del vector.
- Por lo tanto, para cada valor de i, la probabilidad de que el elemento se situé en alguna posición k de las i primeras será de $1/i$.

Ejercicio 1, III

- El número de veces que se repetirá el ciclo WHILE en este caso es (i - k), con lo cual el número promedio de operaciones que se realizan en el ciclo es:

$$\left(\frac{1}{i}\sum_{k=1}^{i} 9(i-k)\right)+4=\frac{9}{2}i-\frac{1}{2}$$

- Por lo tanto, el tiempo de ejecución en el caso promedio es:

$$T(n)=\left(\sum_{i=2}^{n}\left(3+4+\left(\frac{9}{2}i-\frac{1}{2}\right)+3\right)\right)+3=\frac{9}{4}n^2+\frac{47}{4}n-11$$

- Por el modo en que funciona el algoritmo, tales casos van a corresponder a cuando el vector se encuentra ordenado de forma creciente, decreciente o aleatoria.
- Como podemos ver, en este método los órdenes de complejidad de los casos peor, mejor y promedio difieren bastante.
- En el mejor caso, el orden de complejidad resulta ser lineal.

Ejercicio 1, IV

- Mientras que en los casos peor y promedio, la complejidad es cuadrática.
- Este método es muy adecuado para aquellas situaciones donde necesitamos ordenar un vector del que ya conocemos que está casi ordenado, como suele suceder en aquellas aplicaciones de inserción de elementos en bancos de datos previamente ordenados cuya ordenación total se realiza periódicamente.