Tema 1: Eficiencia

Tecnología de la Programación

Índice

Introducción

Objetivo en programación:

Varios algoritmos resuelven el mismo problema ¿cuál elegimos?

1. ¿Qué criterio seguir?
2. ¿Cómo medirlo?

Introducción

Objetivos:

1. Determinar los criterios que definen la eficiencia de un algoritmo
2. Formular una forma de “medir” la eficiencia
3. Caracterizar los problemas que son resolubles en tiempo razonable

Introducción

Objetivos:

1. Determinar los criterios que definen la eficiencia de un algoritmo
   a. Algoritmo fácil de entender, codificar y depurar
   b. Algoritmo que use de forma eficiente los recursos del ordenador
      i. En tiempo de ejecución
      ii. En espacio

Introducción

Factores de los que depende el tiempo de ejecución de un algoritmo/programa

1. Tamaño datos de entrada
2. Contenido datos de entrada
3. El algoritmo en sí
4. La calidad del código generado por el compilador
5. La máquina en la que se ejecute: procesador, lenguaje máquina, …

Analizaremos la eficiencia de los algoritmos de forma independiente a las máquinas

Introducción

Objetivos:

2. Formular una forma de “medir” la eficiencia
    a. Por medio de una función que utiliza el tamaño de los datos como argumento
    b. Esa “función complejidad” da el número de operaciones que requiere la ejecución del algoritmo para una entrada de tamaño dado

$T(n)$ = tiempo de ejecución de un algoritmo con una entrada de tamaño n

Complejidad en mejor caso, en media, en el peor

Notaciones asintóticas

Objetivos: formalizar las unidades de medida

- Son las unidades de medida utilizadas para medir la eficiencia de un algoritmo
  - Se trata de medir el coste en tiempo que tarda en ejecutarse un algoritmo según el tamaño y contenido de los datos de entrada
  - "Asintótico" = eficiencia se estudia para volúmenes grandes de datos
- Coste en tiempo se expresa mediante la **función de complejidad**
  - $f : \mathbb{N} \to \mathbb{R}^+$
  - En la práctica la función de complejidad NO se calcula, solo se estima

Notaciones asintóticas

Objetivos: formalizar las unidades de medida

$$O(f) = \{g : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\} | \exists c_o \in \mathbb{R}^+ \wedge \exists n_0 \in \mathbb{N} \;\; tq \; \forall n \geq n_0, g(n) \leq c_0 f(n)\}$$

Conjunto de funciones que crecen como máximo con la misma rapidez que f.

Si $g \in O(f)$ diremos que **g es del orden de f** o que **g es de O de f**

Notaciones asintóticas

Ejemplos y propiedades:

$$p(n) = a_0 + a_1 n + \ldots + a_k n^k, a_k > 0 \Rightarrow p(n) \in O(n^k)$$

$$f \in O(f)$$

$$f \in O(g) \Rightarrow O(f) \subseteq O(g)$$

$$f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$$

$$O(f) = O(g) \Leftrightarrow f \in O(g) \wedge g \in O(f)$$

$$\forall c \in \mathbb{R}^+ : g \in O(f) \Leftrightarrow cg \in O(f)$$

$$\forall c \in \mathbb{R}^+ : g \in O(f) \Leftrightarrow c + g \in O(f)$$

$$lim_{n \to \infty} f(n)/g(n) = 0 \Rightarrow O(f) \subseteq O(g)$$

Notaciones asintóticas

Operaciones con órdenes de complejidad

$$O(f) + O(g) = \{h : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} | \exists f' \in O(f) \wedge \exists g' \in O(g)\ tq\ h(n) = f'(n) + g'(n) \forall n \in \mathbb{N}\}$$

$$O(f)O(g) = \{h : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} | \exists f' \in O(f) \wedge \exists g' \in O(g)\ tq\ h(n) = f'(n)g'(n) \forall n \in \mathbb{N}\}$$

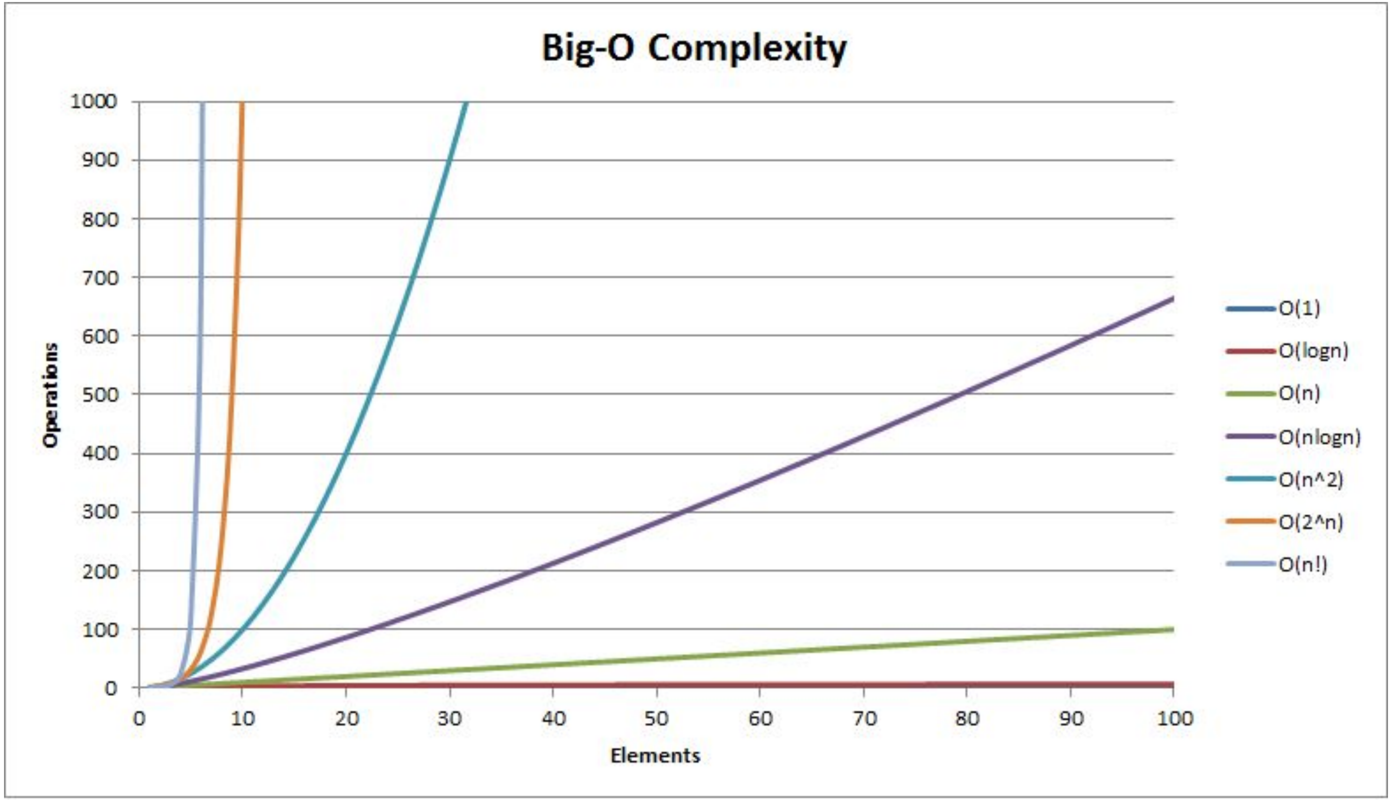
Reglas prácticas

$$O(f) + O(g) = O(f + g) = O(máx\{f, g\})$$

$$O(f)O(g) = O(fg)$$

$$O(1) \subseteq O(log\, n) \subseteq O(n) \subseteq O(n\, log\, n) \subseteq O(n^2) \subseteq \ldots \subseteq O(n^k) \subseteq O(2^n) \subseteq O(n!)$$

Notaciones asintóticas



Notaciones asintóticas

¿Qué significa que un algoritmo sea …

$$O(1)?$$

$$O(n)?$$

$$O(n^2), O(n^3), O(n^4), \ldots?$$

$$O(2^n)?$$

$$O(log\, n)?$$

Notaciones asintóticas

Reglas prácticas para estimar la función de complejidad:

O(1)

- Instrucciones elementales (asignación, lectura, escritura).
- Evaluación de expresiones aritméticas o booleanas
- Acceso a componentes de un vector o a campos de un registro

Notaciones asintóticas

Composición secuencial

S1
S2

$$\left.\begin{array}{l} T(S1) \in O(f_1) \\ T(S2) \in O(f_2) \end{array}\right\} \Rightarrow T\begin{pmatrix} S1 \\ S2 \end{pmatrix} \in O(\text{máx}\{f_1, f_2\})$$

Notaciones asintóticas

Composición condicional

```
si B entonces
        S1
si_no
        S2
fsi
```

$$\left.\begin{array}{l} T(B) \in O(f_B) \\ T(S1) \in O(f_1) \\ T(S2) \in O(f_2) \end{array}\right\} \Rightarrow T(si \ldots fsi) \in O(max\{f_B, f_1, f_2\})$$

Notaciones asintóticas

Composición iterativa

```
mq B hacer
        S
fmq
```

$$\left.\begin{array}{c} numIter \in O(f_{iter}) \\ T(BS) \in O(f_{B,S}) \end{array}\right\} \Rightarrow T(mq \ldots fmq) \in O(f_{B,S}\, f_{iter})$$

Ejemplos

```
función suma_datos(A: tmatriz; n:entero) devuelve entero
variables
    i, j, suma: entero
principio
    suma ← 0
    para i ← 1 hasta n hacer
            para j ← 1 hasta n hacer
                suma ← suma + A[i,j]
            fpara
    fpara
    devuelve(suma)
fin
```

Ejemplos

```
función producto(A,B: tmatriz; n:entero) devuelve tmatriz
variables
     i, j, k: entero
     C: tmatriz
principio
     para i ← 1 hasta n hacer
               para j ← 1 hasta n hacer
                    C[i,j] ← 0
                    para k ← 1 hasta n hacer
                         C[i,j] ← C[i,j] + A[i,k]*B[k,j]
                    fpara
               fpara
     fpara
     devuelve(C)
fin
```

Ejemplos

```
función traspuesta(A: tmatriz; n:entero) devuelve tmatriz
variables
    i, j: entero
    B: tmatriz
principio
    para i ← 1 hasta n hacer
            para j ← 1 hasta n hacer
                B[i,j] ← A[j,i]
            fpara
    fpara
    devuelve(B)
fin
```

Ejemplos

```
acción traspuesta(e/s A: tmatriz; ent n:entero)
variables
    i, j, aux: entero
principio
    para i ← 1 hasta n hacer
            para j ← i hasta n hacer
                aux ← A[i,j]
                A[i,j] ← A[j,i]
                A[j,i] ← aux
            fpara
    fpara
fin
```

Ejemplos

```
acción ejemplo(ent v: tvector; ent n:entero)
variables
    i, j: entero
principio
    para i ← 1 hasta n hacer
        j ← n
        mientras que (j>0) hacer
            v[j] ← v[j] + 1
            j ← j div 2
        fmq
    fpara
    para i ← 1 hasta n hacer
        escribir(v[i])
    fpara
fin
```

Eficiencia

Suponemos que trabajamos con una máquina que tiene por unidad de tiempo un milisegundo y que disponemos de cinco diferentes algoritmos para resolver un mismo problema. La siguiente tabla establece el tamaño máximo que se puede resolver en diferentes tiempos:

Algoritmo	Complejidad	1 segundo	1 minuto	1 hora
A_1	n	1000	6×10^4	3.6×10^6
A_2	n log n	140	4893	2×10^5
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9-10	15-16	21

Eficiencia

A continuación, multiplicamos por mil la velocidad de la máquina (1 microsegundo es igual a 10^6 operaciones por segundo):

Complejidad	n = 20	n = 40	n = 60
n	0.00002 seg	0.00004 seg	0.00006 seg
n^2	0.0004 seg	0.0016 seg	0.0036 seg
n^3	0.008 seg	0.064 seg	0.216 seg
2^n	1 seg	12.7 días	366 siglos

Eficiencia métodos ordenación: selección directa

```
void seleccionDirecta (int v[], int n){
    int i, j, indmenor, aux;
    for(i = 1; i <= n-1; i++){
        indmenor = i;
        for(j = i+1; j <= n; j++){
            if(v[indmenor] > v[j]){
                ...
            }
        }
    }
}
```

Eficiencia métodos ordenación: inserción directa

```
void insercionDirecta (int v[], int n){
    int i, j;
    for(i = 2; i <= n; i++){
        v[0] = v[i];
        j = i;
        while(v[j-1] > v[0]){
                ...
        }
        v[j] = v[0];
    }
}
```

Eficiencia métodos ordenación: burbuja

```
void burbuja (int v[], int n){
    int i, j, aux; // i es el elemento a fijar
    for(i = 1; i <= n-1; i++){
        for(j = n; i+1 <= j; j--){
            if(v[j-1] > v[j]){
                …
            }
        }
    }
}
```

Eficiencia métodos ordenación: heapsort

```
void hundir (int v[], int n, int i){
    int j;
    do{
        j = i; // Busca el hijo menor del nodo i
        if((2*j <= n) && (v[2*j] > v[i]))
            i = 2*j;
        if((2*j < n) && (v[2*j+1] > v[i]))
            i = 2*j+1;
        permutar(v[i],v[j]);
    }while(i!=j);
}
```

Eficiencia métodos ordenación: heapsort

```
void heapSort (int v[], int n){
    int i;
    for(i = n/2; 1 <= i; i--)
        hundir(v, n, i);
    for(i = n; 2 <= i; i--){
        permutar(v[1],v[i]);
        hundir(v, n, i);
    }
}
```

Eficiencia métodos ordenación

	Selección directa	Inserción directa	Burbuja	Heapsort
n=1000	0 seg	0 seg	0 seg	0 seg
n=10000	0.172 seg	0.125 seg	0.39 seg	0 seg
n=50000	4.243 seg	2.793 seg	9.641 seg	0.016 seg
n=100000				
n=500000				
n=1000000				

Otras notaciones asintóticas

Conjunto de funciones que crecen con igual o mayor rapidez que f

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} | \exists c_0 \in \mathbb{R}^+ \wedge \exists n_0 \in \mathbb{N}\, tq\, \forall n \geq n_0, g(n) \geq c_0 f(n)\}$$

Conjunto de funciones de orden exacto de f

$$\Theta(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} | \exists c, d \in \mathbb{R}^+ \wedge \exists n_0 \in \mathbb{N}\, tq\, \forall n \geq n_0, cf(n) \leq g(n) \leq df(n)\}$$

$$\Theta(f) = O(f) \cap \Omega(f)$$

$$lim_{n \to \infty} f(n)/g(n) = k \in \mathbb{R}^+ \Rightarrow \Theta(f) = \Theta(g)$$