Análisis de Algoritmos

Algoritmos (Universidade da Coruña)

ANÁLISIS DE ALGORITMOS

Análisis de eficiencia de los algoritmos

<u>Objetivo:</u> predecir el comportamiento del algoritmo. Aspectos cuantitativos: tiempo de ejecución, cantidad de memoria.

Disponer de una medida de su eficiencia: "teórica" o no exacta -> aproximación suficiente para comparar, clasificar. Acotar T(n): tiempo de ejecución, n = tamaño del problema (a veces de la entrada).

Si n-> ∞ : comportamiento es asintótico. T(n) = O(f(n))

f(n) es una cota superior de T(n) suficientemente ajustada y f(n) crece más deprisa que T(n).

<u>Aproximación:</u> 1. Ignorar factores constantes. 2. Ignorar términos de orden inferior: n + cte -> n

Notaciones asintóticas

<u>Objetivo:</u> Establecer un orden relativo entre las funciones comparando sus tasas de crecimiento.

<u>La notación O:</u> T(n), f(n): Z^+ -> R^+

DEF: T(n) = O(f(n)) si existen constantes c > 0 y n_0 > 0: T(n) </= c*f(n) para todo n >/= n_0

Siendo n_0 umbral. T(n) es O(f(n)), T(n) pertenece a O(f(n)), "la tasa de crecimiento de T(n) </= que la de f(n)" -> f(n) es una cota superior de T(n).

- Reglas prácticas para trabajar con la O:
  - DEFINICIÓN: f(n) es monótona creciente si n_1 >/= n_2 => f(n_1) >/= f(n_2)
  - TEOREMA: ∀c > 0,a > 1,f(n) monotona creciente: $(f(n))^c = O(a^{f(n)})$

Cálculo de los tiempos de ejecución

<u>Modelo de computación:</u>

- Ordenador secuencial
- Instrucción <-> paso (no hay instrucciones complejas)
- Entradas: tipo único ("entero") -> sec(n)
- Memoria infinita + "todo está en memoria"

<u>Alternativas: un paso es...</u>

- Operación elemental: operación cuyo tiempo de ejecución está acotado superiormente por una constante que sólo depende de la implementación.
- Operación principal [Manber]: operación representativa del trabajo del algoritmo: el número de operaciones principales que se ejecutan debe ser proporcional al número total de operaciones.
- La hipótesis de la operación principal supone una aproximación mayor.
- En general, usaremos la hipótesis de la operación elemental.
- En cualquier caso, se ignora: lenguaje de programación, procesador, sistema operativo, carga...
  - Sólo se considera el algoritmo, el tamaño del problema,...

Debilidades:

Operaciones de coste diferente ("todo en memoria => lectura en disco = asignación) -> contar separadamente según tipo de instrucción y luego ponderar == factores== dependiente de la implementación => costos y generalmente inútil.

Faltas de página ignoradas...

Análisis de casos:

Consideramos distintas funciones para T(n):

$T_{mejor}(n)$, $T_{medio}(n)$ -> representativa, más complicada de obtener, $T_{peor}(n)$ -> en general, la más utilizada.

$$T_{mejor}(n) \leq T_{medio}(n) \leq T_{peor}(n)$$

Ordenación por inserción

```
procedimiento Ordenación por Inserción (var T[1..n])
    para i := 2 hasta n hacer
        x := T[i];
        j := i - 1;
        mientras j > 0 y T[j]>x hacer
            T[j+1] := T[j];
            j := j - 1;
        fin mientras;
        T[j+1] := x
    fin para
fin procedimiento
```

Peor caso : "insertar siempre en la primera posición"

Mejor caso: "no insertar nunca"

Ordenación por selección

procedimiento Ordenación por Selección (var T[1..n])

```
para i:=1 hasta n - 1 hacer
    minj := i;
    minx := T[i];
    para j := i + 1 hasta n hacer
        si T[j] < minx entonces
            minj := j;
            minx := T[j];
        fin si
    fin para;
    T[minj]:=T[i];
    T[i]:= minx;
finpara;
```

fin procedimiento

Reglas para calcular O

1. Operación elementl = 1 <-> Modelo de Computación
2. Secuencia: S1 = O(f1(n)) ^ S2 = O(f2(n)) => S1;S2 = O(f1(n) + f2(n)) = O (max(f1(n), f2(n)))
3. Condición: B = O(fb(n)) ^S1 = O(f1(n)) ^ S2 = O (f2(n)) => si B entonces S1 sino S2 = O(max(fb(n), f1(n), f2(n)))
4. Iteración: B;S = O(fb,s(n)) ^ num.iteracion = O(fiter(n))
    a. mientras B hacer S = O(fb,s(n) * fiter(n)) ssi el coste de las iteraciones no varía, sino sumatorio costes individuales.
    b. para i <- x hasta y hacer S = O(fs(n) * num.iter) ssi el coste de las iteraciones no varía, sino sumatorio de costes individuales.
        i. B es comparar 2 enteros = O(1); num.iter = y - x + 1