Eficiencia[1] o coste de algoritmos

Es la medida del consumo de recursos que realiza un algoritmo. Las magnitudes más usadas son:

- Tiempo: número de instrucciones
- Espacio: memoria ocupada (excluyendo los propios datos)

Se mide sin tener en cuenta factores externos, como la máquina, el lenguaje, etc.

1: también se usa “complexity”, pej. ver la doc. de la STL

Eficiencia o coste de algoritmos

- Análisis caso peor: se ha de identificar el valor o la forma de los datos más desfavorables
- Análisis caso medio: se ha de obtener una distribución del espacio de datos y obtener una media estadística de los costes
- Análisis amortizado: se tiene en cuenta el número de ejecuciones previstas

Eficiencia o coste de algoritmos

Se calcula en función del tamaño de los datos. Por ejemplo, si tenemos un algoritmo sobre un vector/lista/árbol de tamaño N, el recuento de instrucciones da lugar a una función *f(N)*.

Si los datos son puramente numéricos, podemos considerar como tamaño el valor, el número de dígitos o el número de bits

Eficiencia o coste de algoritmos

Costes típicos (tamaño de los datos = N):

- *Constante*: independiente de N. *size* de clases de la STL, consulta o modificación de posiciones de vectores o listas, *push*, *top*, *front*, *arrel*, etc. (sin contar las copias)
- *Lineal*: proporcional a N (por ejemplo: 5N+3). Suma de un vector, lista, etc.; búsqueda simple

Eficiencia o coste de algoritmos

Costes típicos (tamaño de los datos = N):

- *Cuadrático*: proporcional a N^2 (ej: $3N^2+5N+7$): métodos sencillos de ordenación de vectores
- Otros costes "polinómicos": N^3, $N^{7/4}$, etc.
- *Exponencial*: proporcional a c^N ($c>1$): *fer i desfer*, permutaciones de N elementos

Eficiencia o coste de algoritmos

Costes típicos (tamaño de los datos = N):

- *Logarítmico*: proporcional a $\log_2 N$. Búsqueda dicotómica, inserción, borrado, etc. en *maps* y *sets*
- *Quasilineal*: proporcional a $N \cdot \log_2 N$. Métodos avanzados de ordenación de vectores

Eficiencia o coste de algoritmos

Atención: no se puede decir que *todo* algoritmo con una función de coste de una categoría menor que la de otro sea *siempre* más rápido que éste:

- los costes mencionados ocultan las constantes que multiplican las potencias de N
- los dos algoritmos pueden tener casos peores distintos

Sí que lo será a partir de un valor determinado de N (la función grande dividida por la pequeña tiende a infinito al crecer N)

Eficiencia o coste de algoritmos

Cómo conseguir algoritmos eficientes:

- Tener una buena idea
- Tener una idea no tan buena y evolucionarla hasta sacarle todo el partido ➔ detectar y eliminar cálculos redundantes
  - Iteración: variables auxiliares para conservar información tras cada vuelta
  - Recursividad: inmersiones de eficiencia para conservar información entre las llamadas

Eficiencia o coste de algoritmos

Cómo conseguir *sistemas* eficientes:

- Asignar prioridades a las funcionalidades, por ejemplo detectar las más usadas
- Decidir cuánto estamos dispuestos a sacrificar en la eficiencia de las menos prioritarias
- Optimizar la eficiencia de las más prioritarias, sin perjudicar más de lo previsto al resto

Ejemplo: el famoso *imax* de Cjt_estudiants