Práctica 1: Eficiencia

Gustavo Rivas Gervilla

UNIVERSIDAD
DE GRANADA

Objetivos

Motivación

Eficiencia Teórica

Eficiencia Empírica

Objetivos

Motivación

Eficiencia Teórica

Eficiencia Empírica

Objetivos

- Aprender a calcular la **eficiencia teórica**.
- Aprender a calcular la **eficiencia empírica**.
- Aprender a **ajustar** la curva de eficiencia teórica a la empírica.

Objetivos

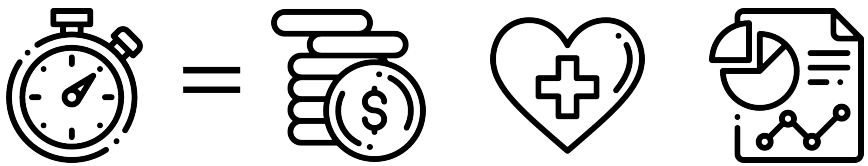
Motivación

Eficiencia Teórica

Eficiencia Empírica

Motivación

La complejidad de un algoritmo (ya sea en **tiempo** o en espacio) es uno de los aspectos cruciales en la evaluación de código.



Como ingenieros nuestro trabajo es buscar siempre la **mejor solución**.

Motivación

Hay problemas de elevada complejidad:

- El problema de encontrar el óptimo (no local) de un problema combinatorio suele ser un problema inabordable $\Longrightarrow$ algoritmos genéticos, enfriamiento simulado, algoritmos de colonia de hormigas...
- El QAP o problema de la asignación cuadrática es uno de estos problemas.
- Determinar el grado de verdad de una fórmula de la aritmética de Presburger es $\mathcal{O}(2^{2^{cn}})$.

Motivación

En las *coding interviews* es importante la eficiencia de las soluciones que propongamos.

- Es parte del libro Cracking The Coding Interview.

- Big O Notation.
- "Performance Matters" by Emery Berger.

También en el campo de la investigación se analiza la eficiencia de los algoritmos:

- SO Kuznetsov y SA Obiedkov. "Algorithms for constructing a set of all concepts of formal context and their Hasse diagrams". En: *Journal of Computer and Systems Sciences International* 1 (2001), págs. 120-129
- Petr Krajca, Jan Outrata y Vilém Vychodil. "Advances in Algorithms Based on CbO.". En: *CLA*. Vol. 672. 2010, págs. 325-337

?

Objetivos

Motivación

Eficiencia Teórica

Eficiencia Empírica

Eficiencia Teórica

algoritmo ≠ implementación

Esta medida es independiente de:

- El *hardware* donde lo ejecutamos.
- El lenguaje donde en el que lo hemos implementado.
- Las bibliotecas empleadas.

Este tipo de medidas son interesantes si tenemos en cuenta las **limitaciones** de los estudios empíricos.

Eficiencia Teórica

Las unidades en las que medimos la eficiencia teórica son las **operaciones elementales**.

operación elemental ≐ operación cuyo tiempo de ejecución se puede acotar por una constante.

- Operaciones aritmético-lógicas.
- Indexaciones[1].
- Asignaciones.
- Incrementos.

[1] Se asume que los acceso a memoria son $\mathcal{O}(1)$.

Eficiencia Teórica

Reglas

Regla de la Suma

Si $t_1 \in f, t_2 \in g \implies t_1 + t_2 \in \mathcal{O}(\max\{f, g\})$.

Regla del Producto

Si $t_1 \in f, t_2 \in g \implies t_1 t_2 \in \mathcal{O}(fg)$.

Eficiencia Teórica

Reglas

Estructura Secuencial

$$\mathcal{O}\left(\sum f_{\text{segmento}}\right)$$

Estructura Condicional

$$\mathcal{O}\left(f_{\text{condición}} + f_{\text{peor camino}}\right)$$

Bucle *for*

$$\mathcal{O}\left(f_{\text{ini}}\right) + \mathcal{O}\left(f_{\text{cond}}\right) + \mathcal{O}\left(f_{\text{ite}}\right)\left[\mathcal{O}\left(f_{body}\right) + \mathcal{O}\left(f_{inc}\right) + \mathcal{O}\left(f_{\text{cond}}\right)\right]$$

Bucle *while*

$$\mathcal{O}\left(f_{\text{cond}}\right) + \mathcal{O}\left(f_{\text{ite}}\right)\left[\mathcal{O}\left(f_{body}\right) + \mathcal{O}\left(f_{\text{cond}}\right)\right]$$

Bucle *do-while*

$$\mathcal{O}\left(f_{\text{ite}}\right)\left[\mathcal{O}\left(f_{body}\right) + \mathcal{O}\left(f_{\text{cond}}\right)\right]$$

Eficiencia Teórica

Suma Matriz

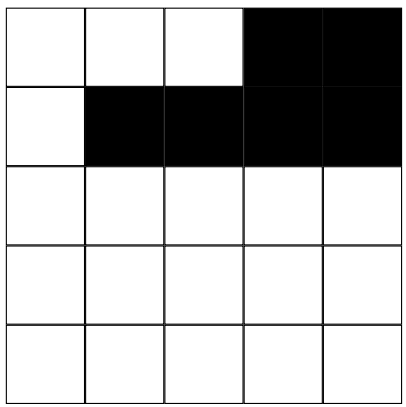
```
1  sum = 0;
2
3  for (i = 0; i < N; i++) {
4      for (j = 0; j < N; j++) {
5          sum += M[i][j];
6      }
7  }
```

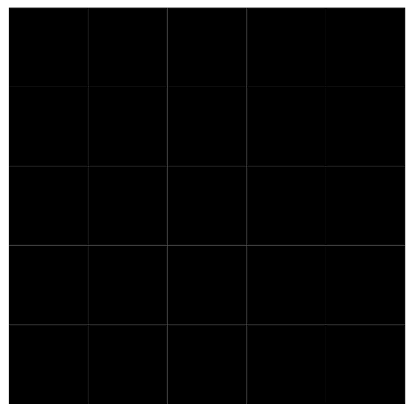
Claramente la eficiencia teórica de este algoritmo es $\mathcal{O}(n^2)$.

?

Eficiencia Teórica

Flood Fill Algorithm





Eficiencia Teórica

Flood Fill Algorithm

```
1   // Los movimientos posibles:
2   int row[] = { -1,-1,-1,0,0,1,1,1 };
3   int col[] = { -1,0,1,-1,1,-1,0,1 };
4
5   // Comprueba si nos podemos mover al pixel (x,y).
6   bool isSafe(char mat[M][N], int x, int y, char target){
7     return (x >= 0 && x < M && y >= 0 && y < N) && mat[x][y] == target;
8   }
9
10  void floodFill(char mat[M][N], int x, int y, char replacement){
11    queue<pair<int, int>> q;
12    q.push({x,y});
13
14    char target = mat[x][y];
15
16    while (!q.empty()){
17          pair<int,int> node = q.front();
18          q.pop();
19
20          int x = node.first , y = node.second;
21
22          mat[x][y] = replacement;
23
24          for (int k = 0; k < 8; k++)
25            if (isSafe(mat, x + row[k], y + col[k], target))
26                  q.push({x + row[k], y + col[k]});
27    }
28  }
```

La eficiencia teórica de este algoritmo es $\mathcal{O}(MN)$.

?

Eficiencia Teórica

Búsqueda Binaria

```
1   function binary_search(A, n, T) is
2       L := 0
3       R := n − 1
4       while L <= R do
5           m := floor((L + R) / 2)
6           if A[m] < T then
7               L := m + 1
8           else if A[m] > T then
9               R := m − 1
10          else:
11              return m
12      return unsuccessful
```

- En el peor caso se dan tantos pasos como sean necesarios para reducir el tamaño del vector a 1.
- ¿Cuántos pasos son necesarios para que esto ocurra dividiendo el vector actual por 2? $\mathcal{O}(\log n)$.

Código obtenido de Wikipedia.

?

Eficiencia Teórica

Recursividad

$$T(n) = \begin{cases} 2T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases}$$

$T(n) = 2T(n-1)+1 = 2^2T(n-2)+2+1 =$
$2^3T(n-3)+2^2+2+1 = \cdots = \sum_{i=0}^{n-1} 2^i = \frac{1-2^n}{1-2} = 2^n - 1 \implies$
$\mathcal{O}(2^n)$.

- Podemos probar la expansión anterior por inducción.
- Hay otras técnicas de resolución de recurrencias[3].

[3] Kimmo Eriksson. "A Summary of Recursion Solving Techniques". En: (1999). URL: https://www.math.kth.se/math/GRU/2012.2013/SF1610/CINTE/mastertheorem.pdf

?

Objetivos

Motivación

Eficiencia Teórica

Eficiencia Empírica

Eficiencia Empírica

El comportamiento real de un algoritmo depende de múltiples factores:

- La entrada.
- La máquina con la que estemos trabajando.
- La implementación del algoritmo (incluyendo el lenguaje de programación empleado).
- Opciones de compilación en el caso de lenguajes compilados.
- ...

Este análisis es tan importante como el análisis de tipo teórico.

Eficiencia Empírica

Estructuras de Datos

Cada estructura de datos tiene sus ventajas e invonvenientes, será nuestro trabajo escoger la forma más adecuada de representar la información para el problema que queremos resolver.

	Acceso	Búsqueda	Insercción	Borrado
Array	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Pila	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Cola	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Lista Enlazada (Simple)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Árbol Binario de Búsqueda	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Árbol Rojo-Negro	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$

Tabla obtenida de artículo.

Eficiencia Empírica

- Dar la eficiencia teórica de un algoritmo es dar una familia de funciones que pertenecen a la clase de equivalencia de la función que modela la eficiencia del algoritmo.
- Estudiando la eficiencia empírica podemos dar un función que modele de forma más precisa la eficiencia del algoritmo.
- Es preferible un algoritmo con una eficiencia $T(n) = n^2$ que uno que siga una $T(n) = 1000n^2 + 2000n + 3000$.

?

Eficiencia Empírica

Midiendo Tiempos

```
1   #include <iostream>
2   #include <ctime>
3   #include <cstdlib>
4   using namespace std;
5
6   int buscar(const int *v, int n, int x){
7     int i = 0;
8     while (i < n && v[i] != x)
9       i = i+1;
10
11    if (i < n)
12      return i;
13    else
14      return -1;
15  }
16
17  void sintaxis(){
18    cerr << "Sintaxis:" << endl;
19    cerr << "  TAM: Tamaño del vector (>0)" << endl;
20    cerr << "  VMAX: Valor máximo (>0)" << endl;
21    cerr << "Genera un vector de TAM número aleatorios en [0,MAX[" << endl;
22    exit(EXIT_FAILURE);
23  }
24
25  int main(int argc, char * argv[]){
26    if (argc != 3)
27      sintaxis();
28    int tam = atoi(argv[1]);
29    int vmax = atoi(argv[2]);
```

Eficiencia Empírica

Midiendo Tiempos

```
30      if (tam <= 0 || vmax <= 0)
31        sintaxis();
32
33      //Generación del vector aleatorio.
34      int *v = new int[tam];
35      srand(time(0));
36      for (int i = 0; i < tam; i++)
37        v[i] = rand() % vmax;
38
39      clock_t tini;
40      tini = clock();
41
42      int x = vmax + 1; //Forzamos el peor caso.
43      buscar(v, tam, x);
44
45      clock_t tfin;
46      tfin = clock();
47
48      cout << tam << "\t" << (tfin - tini) / (double) CLOCKS_PER_SEC  << endl;
49
50      delete [] v;
51    }
```

Eficiencia Empírica

Midiendo Tiempos

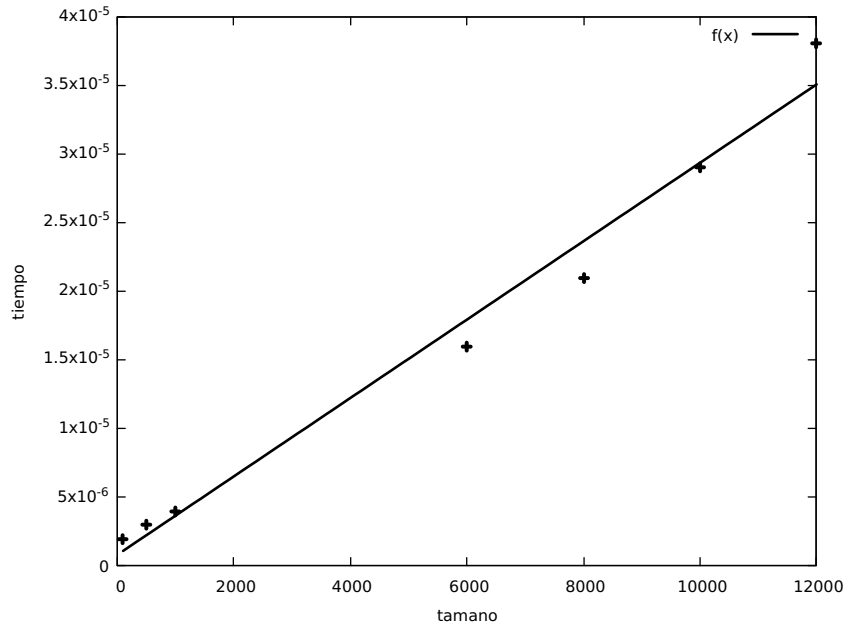
```
1   #!/bin/sh
2   tams=( 100 500 1000 6000 8000 10000 12000 )
3
4   i=$inicio
5   echo > tiempos7.dat
6   for i in "${tams[@]}"
7   do
8       echo Ejecucion tam = $i
9       echo `./busqueda $i 10000` >> tiempos7.dat
10  done
```

```
1
2   100 8e-06
3   500 1.3e-05
4   1000 1.6e-05
5   6000 5.8e-05
6   8000 7.9e-05
7   10000 6.5e-05
8   12000 8.3e-05
```

Eficiencia Empírica

GNUPLOT

```
1  set xlabel "tamano" textcolor rgb "#000000
       "
2  set ylabel "tiempo" textcolor rgb "#000000
       "
3  set xtics textcolor rgb "#000000"
4  set ytics textcolor rgb "#000000"
5  set border lc rgb "#000000"
6  set key textcolor rgb "#000000"
7
8  f(x) = a*x + b
9  fit f(x) "tiempos7.dat" via a, b
10
11 plot "tiempos7.dat" lw 3 lc rgb "#000000"
       notitle, f(x) lc "#000000" lw 2
12 pause -1 "Hit_any_key_to_continue"
```



Para ejecutar el código anterior simplemente ejecutamos en una terminal gnuplot nombreDelFichero.

?

Eficiencia Empírica

GGPLOT

- Otra opción es usar ggplot, con lo que podemos obtener gráficas de una gran calidad y acabado.
- Podemos usarlo en R y en Python.

Eficiencia Empírica

Opciones de g++

Hay un gran número de opciones de optimización:

- -funsafe-math-optimizations: activa distintas optimizaciones para la aritmética de punto flotante asumiendo que los argumentos y resultados son válidos, y saltándose algunos estándares de la IEEE o ANSI, como por ejemplo la siguiente opción.
- -fno-signed-zeros: ignora el signo de los ceros lo que permite simplificar expresiones como $x + 0{,}0$ o $0{,}0 * x$.
- -fprefetch-loop-arrays: precarga la memoria para mejorar el rendimiento de los bucles que acceden a arrays grandes.

Eficiencia Empírica

Opciones de g++

Unas de estas opciones son distintos niveles de optimización que activan distintas opciones de las anteriores:

- -O0: opción por defecto, reduce el tiempo de compilación.
- -O -O1: el compilador trata de reducir el tamaño del código así como su tiempo de ejecución. Realiza optimizaciones que no supongan un tiempo de compilación muy elevado.
- -O2: activa aún más opciones de optimización siempre que no supongan un aumento del tamaño del ejecutable resultante.
- -O3: realiza las mismas optimizaciones que -O2 y además algunas otras que pueden incrementar el tamaño del ejecutable final.
- -Os: sólo realiza optimizaciones que no incrementen el tamaño del ejecutable y además otras optimizaciones pensadas para reducir el tamaño del mismo. Es la opción adecuada para sistemas empotrados.

¿Alguna pregunta?
Buena semana.

Los iconos empleados en estas diapositivas han sido creados por Freepick para `www.flaticon.com`.