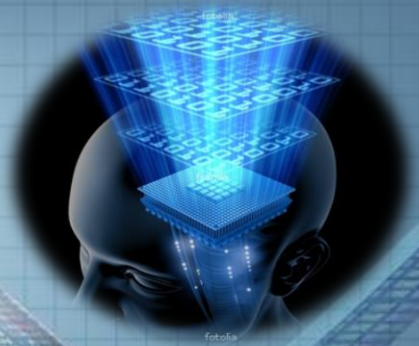
Universidad Autónoma Gabriel Rene Moreno
FICCT
Semestre I/2013

Análisis de Algoritmos (II)



Ejercicio 6
Eficiencia

DEMOSTRAR LA IMPORTANCIA DE LA EFICIENCIA

¿Que utilidad tiene diseñar algoritmos eficientes si las computadoras procesan la información cada vez más rápido?

Bien; para demostrar la importancia de la elaboración de algoritmos eficientes, se plantea el siguiente problema:

Contamos con una computadora capaz de procesar datos en 10^{-4} segundos. En esta computadora se ejecuta un algoritmo que lee registros de una base de datos, dicho algoritmo tiene una complejidad exponencial 2^{n},

¿Cuánto tiempo se tardará en procesar una entrada para N datos?

IMPORTANCIA DE LA EFICIENCIA

n	TIEMPO
10	≈ 1 décima de segundo
20	≈ 2 minutos
30	> 1 día
40	> 3 años
50	= 3 570 años
100	= 4,019,693,684,133,150 milenios

Para 2^N

n	TIEMPO
10	= 1 décima de segundo
20	= 8 décimas de segundo
100	= 1.7 minutos
200	= 13.3 minutos
1000	≈ 1 día

Para N^3

Se puede concluir, que solo un algoritmo eficiente, con un orden de complejidad bajo, puede tratar grandes volumen de datos.

Luego es cierto que:

- Muy eficiente si su complejidad es de orden **log** n
- Eficiente si su complejidad es de orden $N^{\mathbf{a}}$
- Ineficiente si su complejidad es de orden 2^{n}

Se considera que un problema es tratable si existe un algoritmo que lo resuelve con complejidad menor que 2^n , y que es intratable o desprovisto de solución en caso contrario.

Ejercicio 7: Complejidad logarítmica

- En la complejidad logarítmica no se tiene en cuenta la base, porque un cambio entre bases sólo implica multiplicar por una constante.

$$\text{Log}_{10}n = \log_2 n / \log_2 10$$

$$1/\log_2 10 = 0.301$$

$$\text{Log}_{10}n = 0.301 * \log_2 n$$

Complejidad logarítmica

- Es adecuado implementar algoritmos con una complejidad logarítmica, porque los logaritmos crecen muy lentamente aunque el número de datos aumente considerablemente.
- Se define formalmente un logaritmo para cualquier *base (b) ,donde n > 0*, como:

$$\text{Log}_{\mathbf{b}}\, n = k \quad si \quad b^{k} = n$$

- Ej. Un logaritmo de un millón contra el de un billón.

Ejercicio 8
Duplicaciones Repetidas

Empezando con X=1

¿cuántas veces debe ser X duplicado antes de que sea mayor que N?

Empezando en 1 se puede duplicar un valor repetidamente solamente *un número de veces logarítmico antes de alcanzar N*

Nro. de Veces que se duplica	1	2	3	..	P
Valor x	2	4	8	..	N
Valor x	2^1	2^2	2^3	..	2^P

El número de veces que se duplica es p. → $2^p = N$

Es decir p = $\log_2 N$.

Por lo tanto se duplica en un número logarítmico de veces.

Ejercicio 9
Divisiones a la mitad

Empezando con X=N

Si N se divide de forma repetida por la mitad

¿cuántas veces hay que dividir para hacer N menor o igual a 1?

Nro de Veces que se divide	1	2	3	..	P
Nro de datos	N/2	N/4	N/8	..	N/N
Nro de datos	$N/2^1$	$N/2^2$	$N/2^3$	..	$N/2^P$

El número de veces que se divide es p.

$2^p = N$, es decir $p = \log_2 N$.

Por lo tanto se divide N hasta llegar a 1 un número logarítmico de veces.

Ejercicio 10
Las Medianas

```
public boolean Pertenece1 (int [] A, int X)
{
int i;
boolean encontrado = false;
for (i = 0; i < A.length; i++)
      {
      if (A[i] ==X)
            encontrado = true;
      }
return encontrado;
}
```

```
public boolean Pertenece2 (int [] A, int X)
{
int i = 0;
while (i < A.length - 1 && A[i] != X)
            i++;

return A[i] == X;
}
```

¿Número de iteraciones?

Pertenece1 --------- La media es **n.**

Pertenece2 --------- La media es **n/2.**

- Pertenece1 → Realiza N vueltas
- Pertenece2 → Se puede asumir que es la suma de minimo (1) de comparaciones con el maximo de compraciones (n) dividido entre 2
- Luego **(1+N)/2 → ½ * (1+N) = N = O (N)**

Ejercicio 11
Algoritmos de Ordenación

- La ordenación de elementos según un orden ascendente o descendente influye en la velocidad y simplicidad de los algoritmos que los manipulan.

- En general, un conjunto de elementos se almacenan en forma ordenada con el fin de simplificar la recuperación de información manualmente, o facilitar el acceso mecanizado a los datos de una manera más eficiente.

- La complejidad de cualquier algoritmo estima el **tiempo de ejecución** como una ***función del número de elementos a ser ordenados***.

Algoritmos de ordenación

- Cada algoritmo estará compuesto de las siguientes operaciones:
  - COMPARACIONES que prueban si $A_i < A_j$ ó $A_i < B$
    (donde B es una variable auxiliar)
  - INTERCAMBIOS: permutar los contenidos de A_i y A_j ó A_i y B
  - ASIGNACIONES de la forma $B \leftarrow A_i$, $A_j \leftarrow B$ $A_j \leftarrow A_i$

- ***Generalmente, la función de complejidad solo computa COMPARACIONES porque el número de las otras operaciones es como mucho constante del número de comparaciones.***

Ejercicio 12
Ordenación por inserción

- También conocido como **método de la baraja**.
  - Consiste en **tomar elemento a elemento e ir insertando cada elemento en su posición correcta** de manera que se mantiene el orden de los elementos ya ordenados.
- Los jugadores de cartas para ordenar: toman carta por carta manteniendo la ordenación.

Ordenación por Inserción

Se toma el primer elemento, luego se toma el segundo y se inserta en la posición adecuada para que ambos estén ordenados, se toma el tercero y se vuelve a insertar en la posición adecuada para que los tres estén ordenados, y así sucesivamente.

1. Suponemos el primer elemento ordenado.
2. Desde el segundo hasta el último elemento, hacer:
   - Suponer ordenados los **(i–1)**primeros elementos.
   - Tomar el elemento **i**
3. Buscar su posición correcta
4. insertar dicho elemento, obteniendo **i** elementos ordenados

Ordenación por Inserción

```
void ordInsercion (int [] a, int n)
{
	int i, j;
	int aux;
	for (i = 1; i < n; i++)
	{	/* índice j explora la sublista a[i-1]..a[0] buscando la
		   posición correcta del elemento destino, lo asigna a a[j] */
		   j = i;
		   aux = a[i];
		   /* se localiza el punto de inserción explorando hacia abajo */
		   while (j > 0 && aux < a[j-1])
		   {	/* desplazar elementos hacia arriba para hacer espacio */
			    a[j] = a[j-1];
			    j--;
		   }
		   a[j] = aux;
	}
}
```

Ordenación por Inserción

CASO PEOR

Cuando el vector está ordenado inversamente.

86 52 45 20 n=4

52 86 45 20

45 52 86 20

20 45 52 86

En general, para n elementos se realizan (n-1) intercambios y (n-1) comparaciones. Por tanto, $O(n^2)$.

Pasada	NRO. De intercambios	Nro. De comparaciones
1	1	1
2	2	2
3	3	3

TALLER 1

Ordenación por Inserción Binaria

```
public void OrdInsercionBin(int N)
{
for (int i=1; i < N; i++)
{
  Int elem= A[i];
  Int bajo= 0;
  Int alto = (i-1);
  While (bajo <= alto)  // Busca la posición que corresponde almacenar el elemento a ordenar
              {
                Int medio = (alto + bajo)/2;
                If (elem< A[medio])  alto = medio -1;
                Else bajo = medio + 1;
              }
              for(int  j = (i-1); j >= bajo; j--)
               //Se desplazo todos los elementos mayores que el elemento a ordenar una posicion a
                 la derecha
                A [j+1] = A[j];
              A[bajo] = elem;
  }
}
```

Ordenación por Inserción Binaria

- Con la búsqueda binaria se reduce el número de comparaciones desde $O(n^2)$ hasta un $O(n \log n)$.

- Sin embargo, el número de sustituciones (for) requiere un tiempo de ejecución de $O(n^2)$.

- Finalmente:
  - El orden de complejidad no cambia
  - La ordenación por inserción se usa normalmente sólo cuando *n* es pequeño, y en este caso la búsqueda lineal es igual de eficiente que la búsqueda binaria.

EJERCICIO 12

Ordenación por Burbuja

Se basa en el principio de **comparar e intercambiar pares de elementos contiguos hasta que todos estén ordenados**.

- Desde el primer elemento hasta el penúltimo no ordenado comparar cada elemento con su sucesor intercambiar si no están en orden

Si en la primera PASADA se ordenara todo igual tendría que hacer las N pasadas.

```
public void OrdBurbuja()
{
   for(int pasada=0; pasada < N-1; pasada++)
            for (int j=0; j < (N-pasada-1); j++)
                 if (A[j] > A[j+1]) intercambiar(j,j+1);
}
```

Ordenación por Burbuja MEJORADA

Realizar la tabla para 3 elementos: Numero de pasadas vs Numero de intercambios y comparaciones

```
public void BurbujaMejor()
{
      Booleann Ordenado= false;
      Int pasada = 0;
      While ((Ordenado=false) && (pasada < N-1))
      {
            Ordenado= true;
            for (int j=0; j < (N-pasada-1); j++)
                  if (A[j] > A[j+1])
                   { intercambiar(j,j+1);
                     Ordenado= false;
                   }
            pasada ++;
      }
}
```

Ordenación por Burbuja MEJORADA

El tiempo de ejecución de dicho algoritmo viene determinado por el número de comparaciones, en el peor de los casos **$O(n^2)$.**

- COMPARACIONES:
  $(n-1) + (n-2) + \ldots + 3 + 2 + 1 = n(n-1)/2 \rightarrow O(n^2)$

- INTERCAMBIOS:
  $(n-1) + (n-2) + \ldots + 3 + 2 + 1 = n(n-1)/2 \rightarrow O(n^2)$

Ventas Vs Deventajas Ord. Burbuja

- Su principal ventaja es la simplicidad del algoritmo.

- El problema de este algoritmo es que solo compara los elementos contiguos del VECTOR.
- Si el algoritmo comparase primero elementos separados por un amplio intervalo y después se concentra progresivamente en intervalos más pequeños, el proceso sería más eficaz. Esto llevo al desarrollo de ordenación Shell y QuickSort.

Gracias !

No se puede mejorar el running time en fase de implementación sin cambiar el algoritmo !!

Web

http://www.lab.dit.upm.es/~lprg/material/apuntes/o/index.html
http://latecladeescape.com/articulos/1515-que-es-la-complejidad-de-un-algoritmo
http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060024/Lecciones/Capitulo%20II/rbasicas.htm

Texto electrónico

http://www.lcc.uma.es/~av/Libro/CAP1.pdf

Libro Universidad de Málaga
Antonio Vallecillo
Depto.. Lenguajes y Ciencias de la Computación
ETSI Informática, Campus de Teatinos.