Análisis de Algoritmos

Jhonny Felípez Andrade

jrfelizamigo@yahoo.es

Contenido

- **Consideraciones.**
- **Complejidad algorítmica.**
  - **Análisis teórico.**
  - **Ejemplos**

CONSIDERACIONES

Pregunta

¿Dado un problema será posible encontrar más de un algoritmo que sea correcto?

Algoritmo de Fibonacci

```
int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    a = 0; b = 1;
    while (n >= 2) {
      c = a + b;
      a = b; b = c;
      n = n - 1;
    }
    return c;
}
```

Pregunta

¿Cómo determinar si un algoritmo es mejor que otro?

Algunas respuestas

- Fácil de implementar.
- Fácil de entender.
- Fácil de modificar.
- Utiliza menos memoria.
- **Utiliza menos tiempo de ejecución.**

De lo anterior se ocupa el:

Análisis de la complejidad de algoritmos

COMPLEJIDAD ALGORÍTMICA

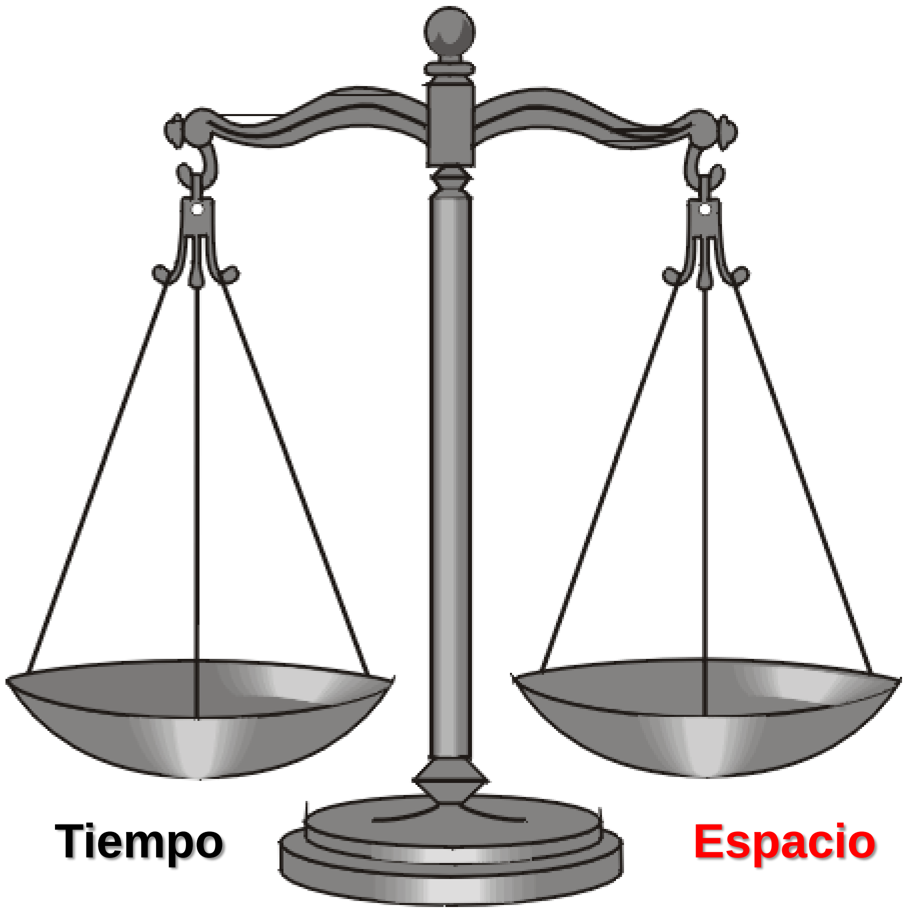
Pregunta

¿Qué es la complejidad algorítmica?

Complejidad de Algoritmos

Es obtener una medida de la cantidad de recursos (espacio o tiempo) que consume determinado algoritmo al resolver un problema.

Complejidad de Algoritmos



Complejidad de Algoritmos. Cont.

Existe la complejidad :

- Temporal: Toma en cuenta el tiempo de ejecución.
- Espacial: Toma en cuenta el espacio de memoria ocupada (no se verá en la materia).

Complejidad de Algoritmos. Cont.

Se tiene dos métodos para obtener la complejidad algorítmica.

- Estimación teórica o a priori.
- Verificación empírica o a posteriori.

Complejidad de Algoritmos. Cont.

El *método teórico* también denominado *a priori*, consiste en determinar matemáticamente la cantidad de recursos necesitados por cada algoritmo como una función cuya variable independiente es el tamaño de los datos del problema.

Complejidad de Algoritmos. Cont.

El *método empírico* también denominado *a posteriori*, consiste en implementar en un computador los algoritmos a comparar, se prueban para distintos tamaños de los datos del problema y se comparan.

ESTIMACIÓN TEÓRICA

Preguntas

Suponiendo que se tiene la siguiente instrucción:

$$x = x + 1$$

1. ¿Cuánto tiempo tarda en ejecutarse esta instrucción?
2. ¿Cuál es el número de veces que ésta instrucción se ejecuta?

Tiempo de Ejecución en Python

tiempo **inicial**

llama función

tiempo **final**

```
import time, math

def sexagesimal_a_radianes(grados):
    return grados * math.pi / 180

t0 = time.time_ns()
sexagesimal_a_radianes(100)
t1 = time.time_ns()
d = t1 - t0
print("Duración ", d, "nano seg.")
```

Tiempo de Ejecución en Java

tiempo **inicial**

llama función

tiempo **final**

```
public class Tiempos {
 static double sexagesimal_a_radianes(double grados) {
  return grados * Math.PI / 180;
 }

 public static void main(String[] args) {
  long t0, t1, d;
  t0 = System.nanoTime();
  sexagesimal_a_radianes(100);
  t1 = System.nanoTime();
  d = t1 - t0;
  System.out.println("Duración " + d + " nano seg.");
 }
}
```

Respuesta 1

Es imposible determinar exactamente cuanto tarda en ejecutar una instrucción

Respuesta 1

A menos que se tenga la siguiente información:

- Características de la máquina.
- Conjunto de instrucciones de la maquina.
- Tiempo que tarda una instrucción de maquina.
- Compilador utilizado.

Assembler de una instrucción

```
d = t1 - t0;
```

```
mov   eax, [t1]
sub   eax, [t0]
mov   [d], eax
```

¿Cuántas instrucciones?

¿Tiempo de ejecución de la instrucción **mov**?

¿Tiempo de ejecución de la instrucción **sub**?

Lo anterior es difícil si se elige una maquina real!.

Respuesta 2

La respuesta a la segunda pregunta se conoce como frecuencia de ejecución.

Respuesta 2

Las consideraciones anteriores nos limitan a obtener solamente la frecuencia de ejecución o la cantidad de veces que se ejecutan todas las instrucciones para un **análisis a priori**.

Análisis a Priori

EJEMPLOS

Ejemplo 1

.

.

$$x = x + 1$$

.

.

La cantidad de frecuencia es uno

Ejemplo 2

```
for i = 1 to n do
    x = x + 1
end
```

La misma sentencia será ejecutada n veces en el programa.

Ejemplo 3

```
for i = 1 to n do
    for j = 1 to n do
        x = x + 1
    end
end
```

n^2 veces (asumiendo $n \geq 1$)

Contando Operaciones de un algoritmo

Suponga que los pasos tienen un **tiempo constante**:

- Operaciones matemáticas
- Comparaciones
- Asignaciones

Luego cuente el número de operaciones ejecutadas en función del tamaño de la entrada.

```
static double
   sexagesimal_a_radianes(double grados) {
  return grados * Math.PI / 180;         3 operaciones
}
static double misuma(int n) {
  int total = 0;                1 operación
  for (int i = 1; i <= n; i++)         3 operaciones
    total = total + i;                 2 operaciones
  return total;                 1 operación
}
```

Ciclo n veces

misuma -> 2 + 5n

Contando Operaciones de un algoritmo

```
static double sexagesimal_a_radianes(double grados) {
  return grados * Math.PI / 180;
}
static double misuma(int n) {
     int total = 0;         1 operación
     int i = 1;             1 operación
     while (i <= n) {       1 operación
         total = total + i;      2 operaciones
         i++;                    1 operación
     }
    return total;           1 operación
}
```

Ciclo n veces

misuma -> 3 + 4n

Cambia un poco el resultado de misuma a pesar de que el algoritmo hace lo mismo, pero, se cambio la implementación.

Comportamiento del algoritmo

Lo bueno es que el recuento es independiente de la computadora en la que se ejecuta, sin considerar el tiempo de ejecución del algoritmo sobre una computadora en particular.

Mas preguntas a responder

¿El algoritmo tiene algún comportamiento particular?

¿Que sucederá si doblo el tamaño de la entrada?

¿Se duplicará o cuadruplicará la cantidad de tiempo que necesito o lo aumentara por un factor de 10?.

Comportamiento del algoritmo

Para responder lo anterior tenemos que sumar ciertamente el número de operaciones de las operaciones. Pero tendríamos que pensar realmente en cuáles serán las operaciones que se tendrán que considerar.

Comportamiento del algoritmo

Además el recuento varía para diferentes valores de entradas. Y podemos usarlo para establecer una relación entre las entradas y el recuento esto reflejará el comportamiento del algoritmo.

Ejemplo 4

Suma los primeros n números naturales.

Este algoritmo tiene un solo argumento y se medirá el comportamiento de este algoritmo en función al tamaño del valor de n.

Suma loe primeros naturales

Cantidad de frecuencia para n > 1

Paso	Algoritmo	Frecuencia
1	`static double misuma(int n) {`	1
2	`    int total = 0;`	1
3	`    for (int i = 1; i <= n; i++)`	n+1
4	`        total = total + i;`	n
5	`    return total;`	1
	`}`	
	t(n) =	2n + 4

Ejemplo 5

Suma los n elementos de un vector.

Este algoritmo tiene un solo argumento y se medirá el comportamiento de este algoritmo en función al tamaño del vector.

Vector suma

Cantidad de frecuencia para n > 1

Paso	Algoritmo	Frecuencia
1	`static double suma_vector(int v[]) {`	1
2	`int n = v.length;`	1
3	`int total = 0;`	1
4	`for (int i = 0; i < n; i++)`	n+1
5	`total = total + v[i];`	n
6	`return total;`	1
	`}`	
	$t(n) =$	$2n + 5$

Ejemplo 6

Búsqueda de un elemento en un vector de n elementos.

Este algoritmo que tiene dos argumentos y se medirá el comportamiento de este algoritmo en función al tamaño del vector no al tamaño del elemento a buscar.

Búsqueda

Cantidad de frecuencia para n > 1

Paso	Algoritmo	Frecuencia
1	`static boolean busqueda(int v[], int e) {`	1
2	`  int n = v.length;`	1
3	`  for (int i = 0; i < n; i++)`	n+1
4	`    if (v[i] == e)`	n
5	`      return true;`	1
6	`  return false;`	1
	`}`	
	t(n) =	2n + 5

- Cuando e es el primer elemento en el vector. Será el mejor de los casos.
- Cuando e no está en el vector. Será el peor de los casos.
- Cuando se revisa aproximadamente hasta la mitad de los elementos del vector. Sera el caso promedio.

Ejercicio de clases

- Plantee y resuelva un ejemplo que tenga
  - el mejor de los casos.
  - el peor de los casos.
  - el caso promedio.

Bibliografía

- Fundamentos de Programación, Jorge Teran Pomier, 2006.
- Fundamentals of Data Structures, Ellis Horowitz, 1981.
- Introduction to Algorithms, third edition, Thomas Cormer, 2009.
- Steven S. Skenia, The Algorithm Design Manual 2da Edición, 2012.

GRACIAS