Estructura de Datos y Algoritmos

ITBA 2024-Q2

Análisis de Algoritmos

Pregunta:

Tengo un cierto problema para resolver, y

2 algoritmos que lo resuelven: **algoA** y **algoB**.

- ¿ Cuál usaríamos?
- ¿ Cómo saber cuál es mejor?

Análisis de Algoritmos

(término introducido por Donald Knuth)

Nos permite "caracterizar" la cantidad de recursos computacionales que usará el mismo cuando se aplique a ciertos datos y evaluar así "su performance".

Nos permite tener una métrica para rankear algoritmos y así poder decidir cuál es mejor y cuál es peor.

Análisis de Algoritmos

La principales métricas para medir la complejidad de algoritmos que ejecutan en máquinas secuenciales (un core) son:

1. El tiempo de ejecución (*runtime analysis/time complexity)*
2. El espacio que utilizan (*space complexity*)

1. Tiempo de ejecución

Pregunta:

¿ Y cómo mido ese tiempo?

1.A) Empíricamente

1.B) Teóricamente

TP 1-Ejer 1

Implementar la clase `MyTimer`
(From scratch)

El intervalo temporal es cerrado-abierto:
[start, end)

start pertenece al intervalo
end no pertenece al intervalo

Interesa saber la cantidad de ms que transcurrieron durante el mismo.

Ej:
[40 ms, 40 ms) => duración 0 ms
[40 ms, 41 ms) => duración 1 ms
[40 ms, 42 ms) => duración 2 ms
[40 ms, 39 ms) => inválido

TP 1-Ejer 1

Implementar la clase `MyTimer`
(From scratch)

- el constructor **MyTimer()** da inicio al mismo.
- el método **stop()** detiene el timer y da fin al intervalo, es decir, dicho valor ya no es parte del mismo.
- El método **toString**() devuelve la duración del intervalo en ms y además el detalle de su duración en días, horas, minutos y segundos con fracción de segundos con 3 decimales.

Tip: Utilizar **`System.currentTimeMillis()`** para obtener el tiempo actual en milisegundos o bien **`System.nanoTime()` y dividir 1000000**

Manejo incorrecto del Timer que no pueda solucionarse: lanzar **RuntimeException**.

TP 1-Ejer 1

Implementar la clase `MyTimer`
(From scratch)

Caso de uso:

```
public static void main(String[] args) {

    MyTimer timer = new MyTimer();
    // bla bla bla …..

    timer.stop();
    System.out.println(timer);
}
```

Salida esperada:

```
(93623040 ms) 1 día 2 hs 0 min 23,040 s
```

Consideraciones

- El problema de API MyTimer anterior es que resulta difícil chequear si funciona correctamente, especialmente cuando transcurren horas e incluso días.

- Extenderemos la API.

TP 1-Ejer 2

Mejorar la clase `MyTimer V2`
(From scratch)

- La forma de iniciar el **MyTimer** V2 es con 2 constructores: sin parámetro (automático) y con parámetro (indicando en ms el comienzo).
- La forma de detener el timer es con 2 **stop**: sin parámetro (automático) y con parámetro (indicando en ms el fin del mismo).
- El método **toString()** devuelve la duración del intervalo en ms y además el detalle de su duración en días, horas, minutos y segundos con fracción de segundos con 3 decimales.

Manejo incorrecto del MyTimer V2 que no pueda solucionarse: lanzar **RuntimeException**. La API es más versátil pero hay más casos para contemplar!!!

TP 1-Ejer 2

Implementar la clase `MyTimer` mejorada
(From scratch)

Casos de uso:

```
MyTimer t1= new MyTimer();
MyTimer t2= new MyTimer(long ms);

// bla bla bla
t1.stop();

// bla bla bla
t2.stop(long ms);

System.out.println(t1);
System.out.println(t2);
```

```
t1= new MyTimer();

// bla bla bla

t1.stop(long ms);

t2= new MyTimer(long ms);

// bla bla bla

t2.stop();
```

Consideraciones

- Si usamos en las aplicaciones bibliotecas propias, crear un "proyecto Java" sirve.
- Pero si usamos bibliotecas externas (otros jars), es complicado mantener actualizaciones y versiones de esta manera, dado que "importamos" jars estáticamente.
- Existe algo muy útil, para los casos en que usamos bibliotecas externas, y por supuesto lo podemos usar aún para aplicaciones nuestras.

Maven

Introducción