DEPARTAMENTO
DE SISTEMAS
INFORMÁTICOS

Diseño de algoritmos
Práctica 0: Medida empírica de la eficiencia de algoritmos

Fernando Cuartero

En casi todo cómputo son posibles una gran variedad de configuraciones para la sucesión de un proceso, y varias consideraciones pueden influir en la selección de estas según el propósito de un motor de cálculos. Una objetivo esencial es escoger la configuración que tienda a minimizar el tiempo necesario para completar el cálculo.
Ada Byron.

1. Introducción

El objetivo de esta primera práctica es doble. En primer lugar, comenzar a familiarizarse con el lenguaje Python, para quien no lo domine, y por otro lado, realizar un enfoque práctico a la comparativa de algoritmos mediante técnicas experimentales. Sin duda, la forma correcta de realizar una comparativa ha de ser mediante técnicas analíticas, sin embargo, eso no siempre será posible, por lo que, de necesitar recurrir al estudio empírico experimental estudiaremos unas nociones elementales.

Para ello, se describe cómo se evalúa el tiempo de ejecución de una función, como se generan los experimentos a analizar y cómo debe realizarse el estudio comparativo. Todo ello, aplicado en lenguaje python.

2. Tiempo de ejecución

Python provee mecanismo de manejo de tiempo en el paquete time, que debe importarse mediante la orden

```
from time import time
```

Con esta importación, utilizaremos la función `time()` que nos proporciona el tiempo transcurrido desde el inicio de la aplicación, en milisegundos. Para medir el tiempo de ejecución del procedimiento `Proc` mediante la variable de tipo entero `t` se deberá hacer

```
t=time()
Proc
t=time()-t
```

3. Algoritmos a evaluar

El archivo `sorting.py` contiene las funciones que implementan los métodos de ordanación directa más conocidos, `Inserción`, `Selección` y `Burbuja`. Se deberán utilizar estas versiones para que la comparativa sea similar en todos. Los algoritmos son los siguientes:

```
def Burbuja(a,n):
for i in range(1,n):
        for j in range(0,n-i):
                if(a[j] > a[j+1]):
                        k = a[j+1]
                        a[j+1] = a[j]
                        a[j] = k;

def Insercion(a,n):
for i in range(1,n):
        v=a[i]
        j=i-1
        while j >= 0 and a[j] > v:
                a[j+1] = a[j]
                j=j-1
        a[j+1]=v

def Seleccion(a,n):
for i in range(0,n-1):
        min=i
        for j in range(i+1,n):
                if a[min] > a[j]:
                        min=j
        aux=a[min]
        a[min]=a[i]
        a[i]=aux
```

3.1. Experimentos a realizar

Para cada algoritmo habrá que realizar experimentos para tres casos, para arrays que estén en orden directo, en orden inverso, o en estado aleatorio. Y para cada casuística se realizarán estudios para diferentes tamaños, desde 100 elementos hasta 2000 con incremento de 100 en 100. Es decir, un total de 20 mediciones para cada caso.

Para los casos de orden directo o inverso, bastará una medida única, pues el vector será el mismo para todos los experimentos y no habrá influencias anómalas. No obstante, en el caso aleatorio puede ser que el ejemplar generado influya en unos casos y en otros no, por lo que para evitar o minimizar errores de medición, lo que se hará es repetir cada experimento 10 veces, mediremos el tiempo en cada caso, y se tomará el promedio. Eso sí, es importante que el vector a ordenar sea siempre el mismo para los 3 algoritmos a analizar.

La generación de números aleatorios se hará usando el paquete `random` y la función `randrange(0,n)` que nos genera un valor aleatorio en el intervalo especificado.

En resumen:

- Deben obtenerse resultados para los tres algoritmos directos, inserción, selección y burbuja.
- Para cada uno de ellos, deben obtenerse resultados para vectores de tamaño 100 a 2000 con incrementos de 100 en 100.
- Los vectores deben estar en tres presentaciones distintas, en orden directo, en orden inverso, o aleatorios.
- En el caso particular de vectores aleatorios, para cada uno de ellos, deben realizarse 10 experimentos distintos, con generaciones distintas, tomando la media. Los vectores generados en cada caso deben ser los mismos para los tres algoritmos.

3.2. Presentación de resultados

La salida de la aplicación a desarrollar deberá contener:

1. Tablas de los resultados, para los tres métodos estudiados en sus tren situaciones, ordenados, inverso y aleatorio, donde ene el eje de las X aparezca el tamaño del vector (n) y en el de las Y aparezca el tiempo $T(n)$.

2. representación gráfica de las funciones respectivas. Puede usarse para ello la librería `matplotlib.pyplot`.

3. Ajuste a función analítica. En este caso, puede hacerse debido a que sabemos que las respectivas funciones de tiempo de ejecución respecto al tamaño del problema son de tipo cuadrático, es decir

$$T(n) = A \cdot n^2 + B \cdot n + C$$

   Por ello, es posible ajustar los valores de A, B, C mediante métodos numéricos. En particular, Python incorpora en su librería matemática `numpy` la función `polyfit` que nos proporciona un ajuste a una función polinómica, como ocurre en este caso, de grado 2. Por ello la tercera salida debe ser un ajuste a la función cuadrática de cada uno de los resultados experimentales.

4. Entregables, evaluación y normas

Para esta práctica, la entrega consiste en un archivo `.zip` que contenga los archivos de código y uno o varios archivos con las salidas y una explicación de los mismos, junto a las conclusiones.

Criterios de evaluación:

- Cualquier práctica cuyo código no sea ejecutable se considerará no entregada. El hecho de que funcione es requisito para que sea correcta, pero puede no ser suficiente.
- El plazo para la entrega de la práctica concluye el día indicado en el entregable de la plataforma. Todos aquellos que entreguen después, serán penalizados multiplicando la nota por un factor 0,8.

Normas de entrega

- La práctica puede desarrollarse de forma individual o por parejas.
- La copia o plagio se penalizará con el suspenso de la asignatura.