ESTRUCTURA DE DATOS. PRACTICA 1

EJEMPLO DE CALCULO DE EFICIENCIA EMPIRICA

DANIEL ALCONCHEL VÁZQUEZ

Aprovechando que hemos hecho el análisis teórico de los algortimos del Reto1, comprobemos algunos de los resultados mediante un análisis empírico.

El algoritmo en cuestión es el siguiente:

```
void eficiencia1(int n){
    int x = 0;
    int i,j,k;

    for(i=1; i<=n; i+=4)
        for(j=1; j<=n; j+0[n/4])
            for(k=1; k<=n; k*=2)
                x++;
}
```

A continuación, escribiremos un pequeño programa donde poder ejecutar el algoritmo:

```
#include<iostream>
#include<ctime> // Recursos para medir tiempos

using namespace std;

void eficiencia1(int n){
    int x = 0;
    int i,j,k;

    for(i=1; i<=n; i+=4)
        for(j=1; j<=n; j+=n/4)
            for(k=1; k<=n; k*=2)
                x++;
}

void sintaxis() {
    cerr<<"Sintaxis:"<<endl;
    cerr<<"  Número Entero (>0)"<<endl;
    exit(EXIT_FAILURE);
}

int main(int argc,char*argv[]) {
    if(argc!=2)               // Lectura de parámetros
        sintaxis();
    int niteraciones = atoi(argv[1]);   // Número de filas
    if(niteraciones<=0)
        sintaxis();
```

```
    clock_t tini;   // Anotamos el tiempo de inicio
    tini=clock();

    eficiencia1(niteraciones);

    clock_t tfin;   // Anotamos el tiempo de finalización
    tfin=clock();   // Mostramos resultados (Tamaño del vector y tiempo de
ejecución en seg.)

    cout<< niteraciones <<"\t"<< tfin-tini <<endl;
}
```

Para dejar el programa compilado, simplemente ejecutamos la siguiente línea de código:

```
$ g++ eficiencia1.cpp -o eficiencia
```

Eficiencia Teórica:

Comenzamos analizando la eficiencia teórica del algoritmo.

Todas las asignaciones y condiciones de los bucles for son, claramente, O(1), por lo que nos centramos en la estructura debucles anidados.

- El primer bucle, claramente, se ejecuta n veces, luego su eficiencia es O(n)
- El segundo bucle solo se ejecuta unas 4 veces, por lo que su eficiencia es O(4)
- En el tercer bucle el contador aumenta exponencialmente, lo que produce que el bucle se ejecute $log_2(n)$, por lo que la eficiencia es O(log(n))

Como los bucles están anidados, por la regla del producto, la eficiencia del algoritmo será O(nlog(n)).

Eficiencia Práctica:

Para analizar la eficiencia práctica usaré un script de shell que ejecuta el programa una serie de veces, tomando los datos que hacen que el programa se ejecute en el peor de los casos. La salida, que consta del número de datos usados y el tiempo de ejecución, se almacenarán en un fichero, para su posterior uso.

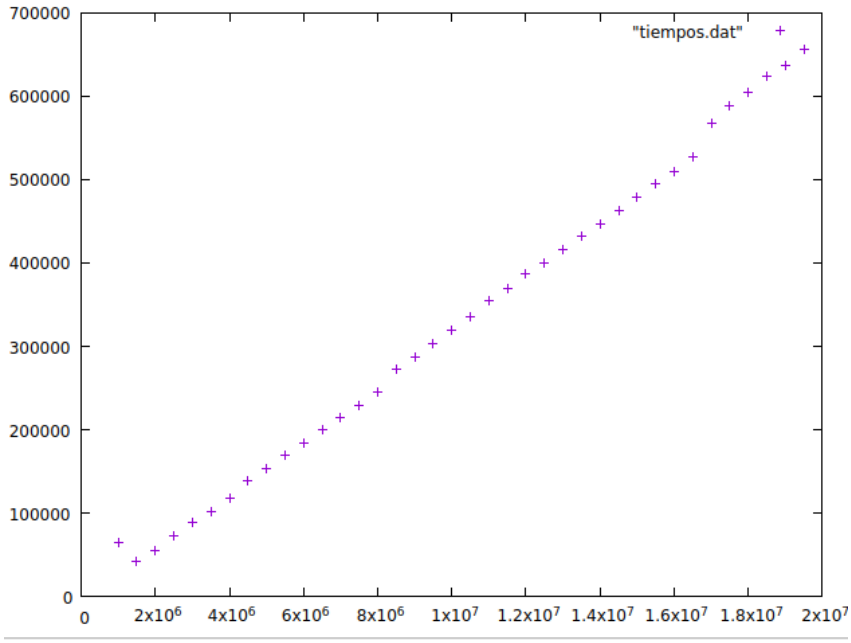
```
#!/bin/bash
for ((i = 1000000 ; i < 1000000000 ; i+=500000)); do
    echo $i
    echo `./eficiencia1 $i` >> tiempos.dat
done
```

Ejecutamos la script y la dejamos trabajando un rato. De esta forma generaremos el fichero de salida (tiempos.dat).

A continuación, abrimos el gnuplot y representamos el fichero de datos usando el siguiente comando:

```
plot "tiempos.dat"
```

Podemos obeservar la siguiente gráfica:

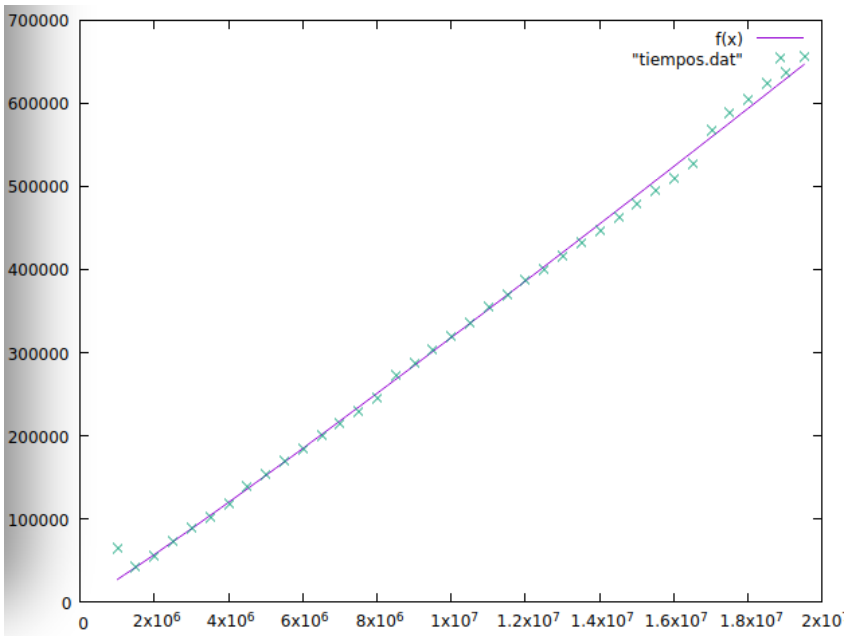


Ahora, escribimos los siguientes comandos:

```
> f(x)=a*x*log(x)+b
> fit f(x) "tiempos.dat" via a, b

a               = 0.00197349
b               = 1
```

Veamos el resultado del ajuste:



Como podemos observar, el ajuste es bastante bueno, por lo que nuestro análisis teórico parece correcto.

Analicemos también el segundo algoritmo del reto:

```
int eficiencia2(bool existe, int n){
    int sum2=0;
```

```
    int k,j;

    if(existe)
        for(k=1; k<=n; k*=2)
            for(j=1; j<=k; j++)
                sum2++;
    else
        for(k=1; k<=n; k*=2)
            for(j=1; j<=n; j++)
                sum2++;
    return sum2;
}
```

Al igual que en el caso anterior, creamos un código donde poder probar el algoritmo:

```
#include<iostream>
#include<ctime> // Recursos para medir tiempos

using namespace std;

int eficiencia2(bool existe, int n){
    int sum2=0;
    int k,j;

    if(existe)
        for(k=1; k<=n; k*=2)
            for(j=1; j<=k; j++)
                sum2++;
    else
        for(k=1; k<=n; k*=2)
            for(j=1; j<=n; j++)
                sum2++;
    return sum2;
}

void sintaxis() {
    cerr<<"Sintaxis:"<<endl;
    cerr<<"  Número Entero (>0)"<<endl;
    cerr<<"  Booleano (True o False)" <<endl;
    exit(EXIT_FAILURE);
}

int main(int argc,char*argv[]) {
    if(argc!=3)               // Lectura de parámetros
        sintaxis();
    int niteraciones = atoi(argv[1]);   // Número de filas
    bool condicion = argv[2];   //  Bool
    if(niteraciones<=0)
        sintaxis();

    clock_t tini;   // Anotamos el tiempo de inicio
    tini=clock();

    eficiencia2(condicion,niteraciones);

    clock_t tfin;   // Anotamos el tiempo de finalización
```

```
    tfin=clock();   // Mostramos resultados (Tamaño del vector y tiempo de
ejecución en seg.)

    cout<< niteraciones <<"\t"<< tfin-tini <<endl;
}
```

Eficiencia Teórica:

Comenzamos analizando la eficiencia teórica del algoritmo.

Todas las asignaciones y condiciones de los bucles for son O(1), por lo que nuevamente nos centramos en los bucles anidados:

Consideración: Al tener dos bloques independientes, la eficiencia del programa equivale a la del bloque de mayor complejidad

Si observamos, tanto en la parte del if como del else, el primer bucle es el mismo (además calculamos su eficiencia en el ejercicio anterior) y su eficiencia es O(log(n)), pero, con un simple vistazo podemos ver que el segundo bucle del bloque else se ejecuta más veces que el del bloque if. Esto nos sugiere que debemos centrarnos en el bloque else, ya que su complejidad será mayor.

- El primer bucle es O(log(n))
- El segundo bucle se ejecuta n veces, luego su eficiencia es O(n)

Ya que los bucles están anidados, usamos la regla del producto, por lo que la eficiencia resultante (que es la misma que la del algoritmo) es O(nlog(n)).

Eficiencia Práctica:

Para analizar la eficiencia práctica usaré un script de shell que ejecuta el programa una serie de veces, tomando los datos que hacen que el programa se ejecute en el peor de los casos. La salida, que consta del número de datos usados y el tiempo de ejecución, se almacenarán en un fichero, para su posterior uso.

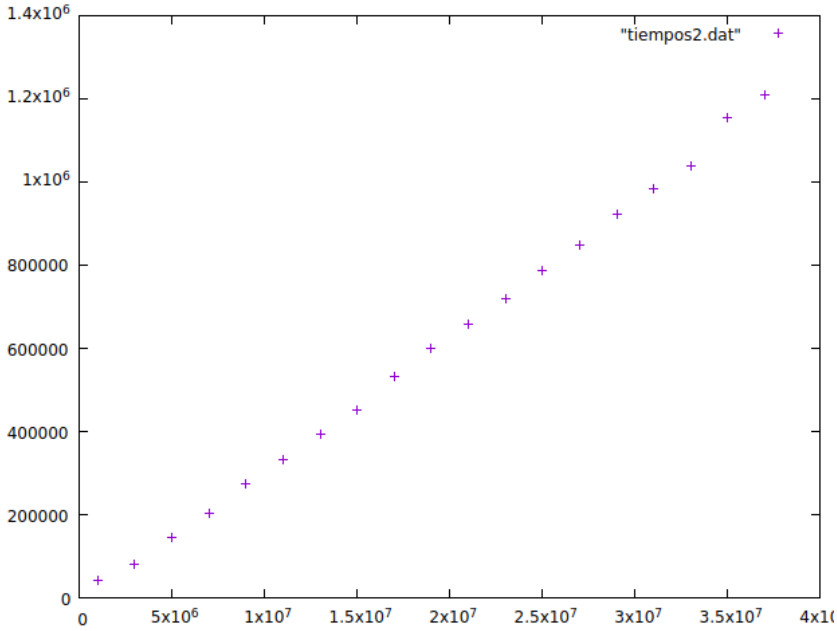
```
#!/bin/bash
for ((i = 1000000 ; i < 1000000000 ; i+=500000)); do
    echo $i
    echo `./eficiencia2 $i false` >> tiempos.dat
done
```

Ejecutamos la script y la dejamos trabajando un rato. De esta forma generaremos el fichero de salida (tiempos2.dat).

A continuación, abrimos el gnuplot y representamos el fichero de datos usando el siguiente comando:

```
plot "tiempos2.dat"
```

Podemos obeservar la siguiente gráfica:

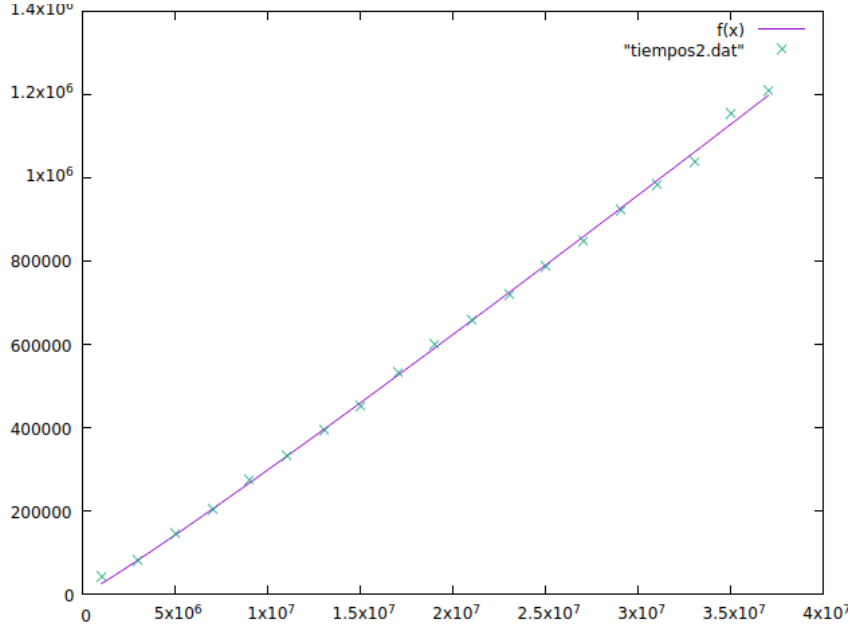


Ahora, escribimos los siguientes comandos:

```
> f(x)=a*x*log(x)+b
> fit f(x) "tiempos.dat" via a, b

a               = 0.00185576
b               = 1
```

Veamos el resultado del ajuste:



Luego, nuestro ajuste parece correcto.