Algorítmica

Grado en Ingeniería Informática

Ejercicio de clase

Búsqueda ternaria

Autores
Carlos Sánchez Páez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2017-2018

Índice

Índice de cuadros

Índice de figuras

1. Enunciado

Realizar un estudio empírico para determinar si es preferible utilizar la búsqueda binaria o la búsqueda ternaria comentada en clase (ambos algoritmos son de orden logarítmico, pero sus constantes ocultas son diferentes).

2. Resolución

2.1. Metodología

Para resolver el ejercicio, ejecutaremos 25 veces cada código con tamaños de problema ascendentes mediante un script. Después, estudiaremos empíricamente su eficiencia y hallaremos el valor de sus constantes ocultas (eficiencia híbrida). Dentro del código fuente, cada algoritmo se ejecuta 1000 veces y se calcula el tiempo medio. Éste procedimiento se realiza ya que los tiempos son tan pequeños que cualquier mínima variación en la carga del sistema causada por otro proceso causa un gran efecto en el tiempo de ejecución, resultando en un gran pico en la representación gráfica. Igualmente, como los tiempos son tan pequeños, he tenido que utilizar el reloj que proporciona más precisión (*high resolution clock*, de la biblioteca *chrono*).

Algoritmo	**Tamaño inicial**	**Tamaño final**	**Incremento**
Búsqueda Binaria Búsqueda Ternaria	50.000.000	530.000.000	20.000.000

Cuadro 1: Tamaños para la ejecución

2.2. Resultados obtenidos

Tamaño del problema	Búsqueda Binaria	Búsqueda Ternaria
50.000.000	$1.02815 \cdot 10^{-7}$	$9.7679 \cdot 10^{-8}$
70.000.000	$9.9185 \cdot 10^{-8}$	$1.02817 \cdot 10^{-7}$
90.000.000	$1.87658 \cdot 10^{-7}$	$9.4667 \cdot 10^{-8}$
110.000.000	$9.7568 \cdot 10^{-8}$	$9.9846 \cdot 10^{-8}$
130.000.000	$1.11688 \cdot 10^{-7}$	$1.0234 \cdot 10^{-7}$
150.000.000	$1.09648 \cdot 10^{-7}$	$1.25571 \cdot 10^{-7}$
170.000.000	$9.5796 \cdot 10^{-8}$	$1.01072 \cdot 10^{-7}$
190.000.000	$1.1167 \cdot 10^{-7}$	$1.23716 \cdot 10^{-7}$
210.000.000	$1.08572 \cdot 10^{-7}$	$1.04633 \cdot 10^{-7}$
230.000.000	$9.6589 \cdot 10^{-8}$	$1.33206 \cdot 10^{-7}$
250.000.000	$1.10112 \cdot 10^{-7}$	$1.63717 \cdot 10^{-7}$
270.000.000	$1.08413 \cdot 10^{-7}$	$1.04575 \cdot 10^{-7}$
290.000.000	$1.12937 \cdot 10^{-7}$	$1.50177 \cdot 10^{-7}$
310.000.000	$1.11755 \cdot 10^{-7}$	$1.10322 \cdot 10^{-7}$
330.000.000	$1.07642 \cdot 10^{-7}$	$1.0309 \cdot 10^{-7}$
350.000.000	$1.07934 \cdot 10^{-7}$	$1.11627 \cdot 10^{-7}$
370.000.000	$1.04718 \cdot 10^{-7}$	$1.09216 \cdot 10^{-7}$
390.000.000	$1.04261 \cdot 10^{-7}$	$1.23385 \cdot 10^{-7}$
410.000.000	$1.7052 \cdot 10^{-7}$	$1.02349 \cdot 10^{-7}$
430.000.000	$1.20473 \cdot 10^{-7}$	$1.15399 \cdot 10^{-7}$
450.000.000	$1.21743 \cdot 10^{-7}$	$1.07027 \cdot 10^{-7}$
470.000.000	$1.24807 \cdot 10^{-7}$	$1.16637 \cdot 10^{-7}$
490.000.000	$1.45116 \cdot 10^{-7}$	$1.1538 \cdot 10^{-7}$
510.000.000	$1.09658 \cdot 10^{-7}$	$1.61301 \cdot 10^{-7}$
530.000.000	$1.19847 \cdot 10^{-7}$	$1.098 \cdot 10^{-7}$
Media	$1{,}144 \cdot 10^{-7}$	$1{,}507 \cdot 10^{-7}$

Cuadro 2: Tiempos obtenidos (seg)

2.3. Cálculo de la constante oculta

Realizamos una regresión mediante *gnuplot* para averiguar la constante:

Algoritmo	Valor de la constante oculta	Porcentaje de error
Búsqueda Binaria	$6.00559 \cdot 10^{-9}$	3.729 %
Búsqueda Ternaria	$5.98818 \cdot 10^{-9}$	3.074 %

Cuadro 3: Bondad del ajuste

2.4. Gráficas

Para apreciar mejor la curvatura de la representación, los ejes están expresados en escala logarítmica.

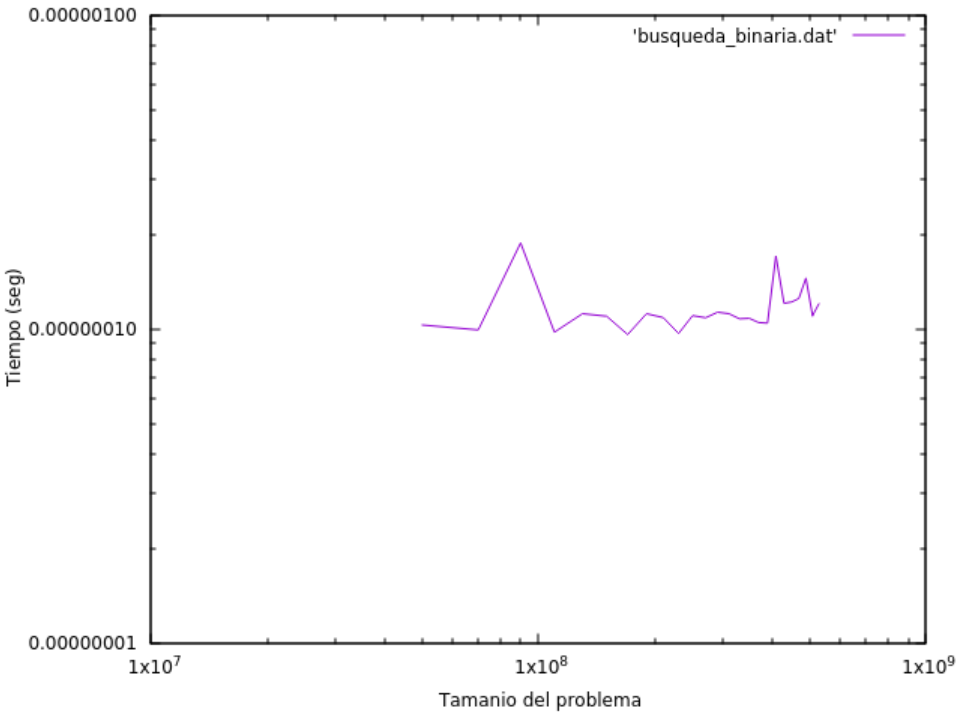

Figura 1: Eficiencia empírica. Búsqueda binaria

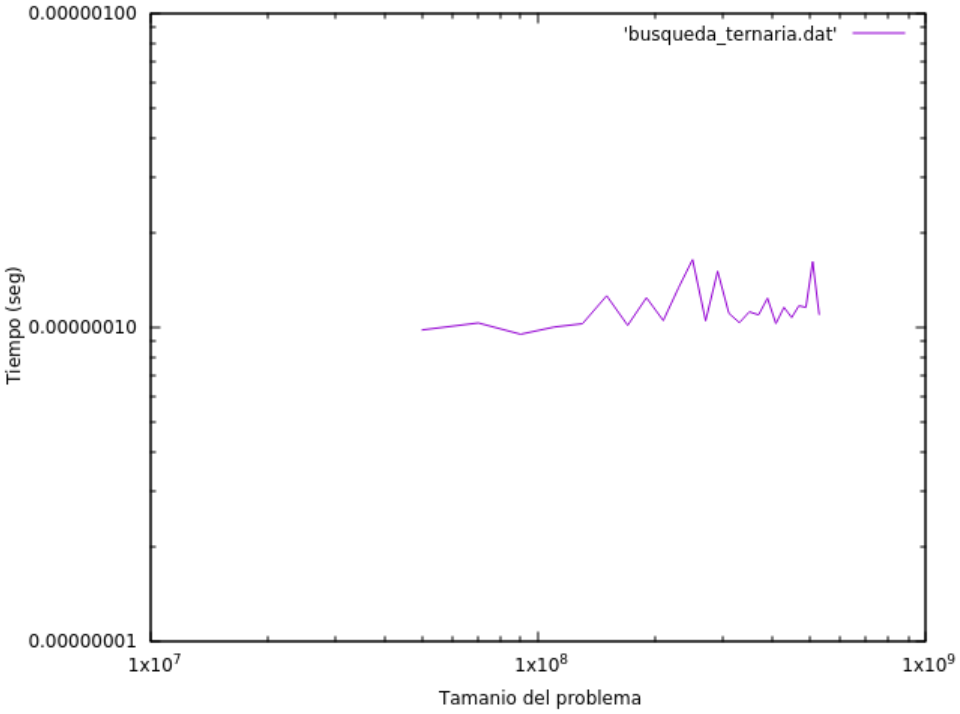

Figura 2: Eficiencia empírica. Búsqueda ternaria

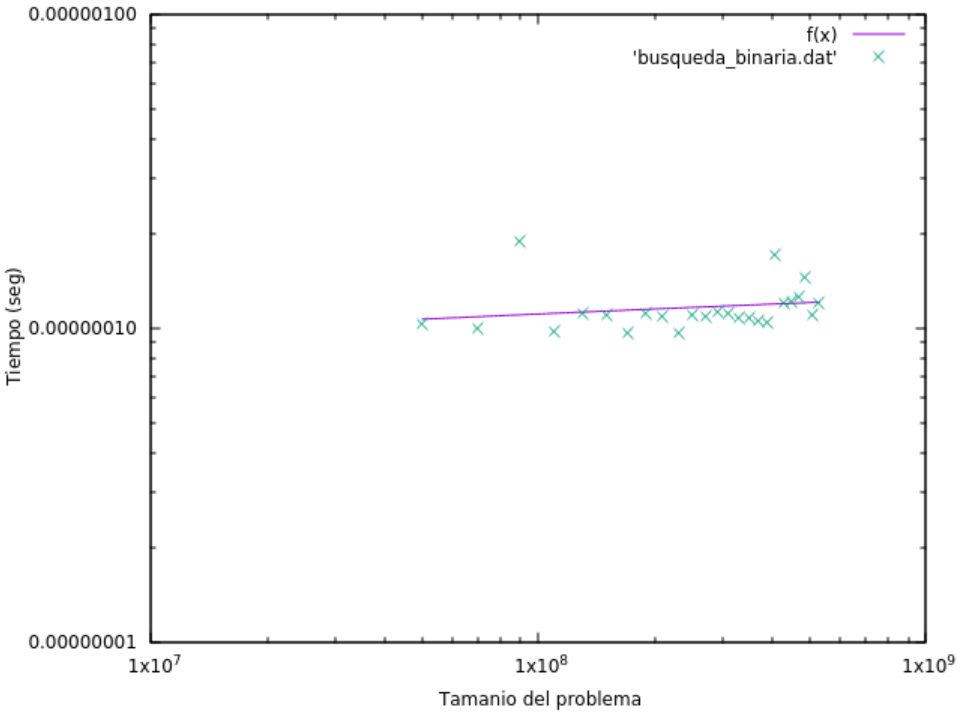

Figura 3: Eficiencia híbrida. Búsqueda binaria

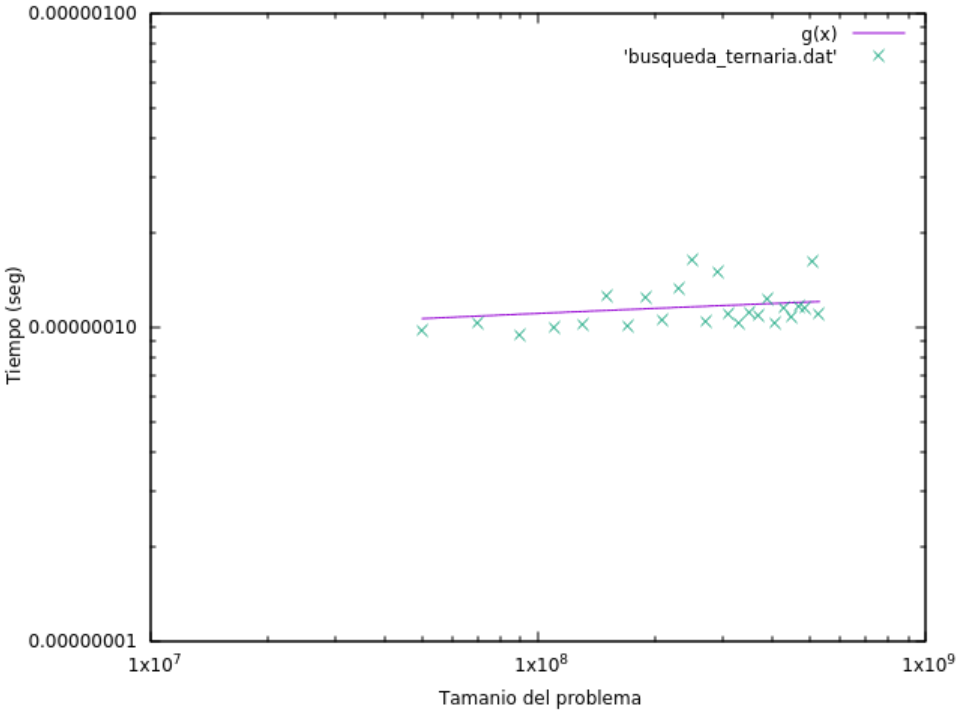

Figura 4: Eficiencia híbrida. Búsqueda ternaria

2.5. Conclusiones

Tanto la búsqueda binaria como la ternaria tienen el mismo orden de eficiencia teórica ($O(\log n)$). Por tanto, la diferencia radicará en sus constantes ocultas.
Según la regresión realizada, la búsqueda ternaria tiene una constante oculta ligeramente menor a la binaria ($2 \cdot 10^{-11}$) Sin embargo, vemos como el ajuste de la binaria es peor (el porcentaje de error es un 0,7 % más). Si nos vamos a los tiempos empíricos, podemos confirmar que la búsqueda ternaria **no es más rápida** que la binaria, por lo que el nuevo algoritmo no nos proporciona ninguna mejora con respecto al tradicional.

2.6. Anexo:Algoritmos desarrollados

2.6.1. Búsqueda binaria

```
#define TEST 0        //Imprimir o no el resultado
#include <iostream>
#include <chrono>
#include <ctime>
#include <ratio>
#include <chrono>
using namespace std;
using namespace std::chrono;

int busquedaBinaria(const int *v, const int i, const int j, const int buscado) {
        //Caso base: nos cruzamos
        if (i > j) {
                return -1;
        }
        else {
                int mitad = (i + j) / 2;
                if (v[mitad] == buscado)                //Acertamos
                        return mitad;
                else if (buscado < v[mitad])        //Buscamos hacia la izquierda
                        return busquedaBinaria(v, i, mitad - 1, buscado);
                else                                                    //Buscamos hacia la derecha
                        return busquedaBinaria(v, mitad + 1, j, buscado);
        }
}

int main(int argc, char **argv) {
        if (argc != 2) {
                cerr << "Uso del programa: " << argv[0] << " <tamaño>" << endl;
                exit(-1);
        }
        int tam = atoi(argv[1]);
        int *v = new int[tam];
        high_resolution_clock::time_point tantes;
        high_resolution_clock::time_point tdespues;
        duration<double> tiempo;
        double acumulado = 0;

        //Inicializar vector con valores aleatorios
        srand (time(NULL));

        for (int i = 0; i < tam; i++)
                v[i] = rand() ;
        // Peor caso: no está
        //Ejecutamos 1000 veces y obtenemos el tiempo medio
        for (int i = 0; i < 1000; i++) {
                tantes = high_resolution_clock::now();
                int pos = busquedaBinaria(v, 0, tam, rand() + 0.5);
                tdespues = high_resolution_clock::now();
                tiempo = duration_cast<duration<double>>(tdespues - tantes);
                acumulado += tiempo.count();
        }
        acumulado /= 1000;


```

```
55  #if TEST
56          cout << "Posición: " << pos << endl;
57  #endif
58          cout << tam << "\t\t" << acumulado << endl;
59
60          delete []v;
61  }
```

2.6.2. Búsqueda ternaria

```
#define TEST 0        //Imprimir o no el resultado
#include <iostream>
#include <chrono>
#include <ctime>
#include <ratio>
#include <chrono>
using namespace std;
using namespace std::chrono;

int busquedaTernaria(const int *v, const int i, const int j, const int buscado) {
        //Caso base: nos cruzamos
        if (i > j)
                return -1;
        else {
                int tam = j - i;
                int tercio = i + (tam / 3);
                int dostercio = i + (tam * 2) / 3;
                if (v[tercio] == buscado) //Acertamos
                        return tercio;
                else if (buscado < v[tercio]) //Primer tercio
                        busquedaTernaria(v, i, tercio - 1, buscado);
                else {
                        if (v[dostercio] == buscado)        //Acertamos
                                return dostercio;
                        else if (buscado > v[dostercio]) //Tercer tercio
                                busquedaTernaria(v, dostercio + 1, j, buscado);
                        else //Segundo tercio
                                busquedaTernaria(v, tercio + 1, dostercio - 1, buscado);
                }
        }
}

int main(int argc, char **argv) {
        if (argc != 2) {
                cerr << "Uso del programa: " << argv[0] << " <tamaño>" << endl;
                exit(-1);
        }
        int tam = atoi(argv[1]);
        int *v = new int[tam];
        high_resolution_clock::time_point tantes;
        high_resolution_clock::time_point tdespues;
        duration<double> tiempo;
        double acumulado = 0;

        //Inicializar vector con valores aleatorios
        srand (time(NULL));

        for (int i = 0; i < tam; i++)
                v[i] = rand() ;
        // Peor caso: no está
        //Ejecutamos 1000 veces y obtenemos el tiempo medio
        for (int i = 0; i < 1000; i++) {
                tantes = high_resolution_clock::now();
                int pos = busquedaTernaria(v, 0, tam, rand() + 0.5);
                tdespues = high_resolution_clock::now();
                tiempo = duration_cast<duration<double>>(tdespues - tantes);
```

```
57                  acumulado += tiempo.count();
58          }
59          acumulado /= 1000;
60
61
62  #if TEST
63          cout << "Posición: " << pos << endl;
64  #endif
65          cout << tam << "\t\t" << acumulado << endl;
66
67          delete []v;
68  }
```

2.6.3. Script para múltiples ejecuciones

```
1   #!/bin/bash
2   if [ $# -eq 3 ]
3   then
4           i="0"
5           tam=$2
6           #Primer argumento: programa a ejecutar
7           #Segundo argumento: tamaño inicial
8           #Tercer argumento : incremento
9           while [ $i -lt 25 ]
10          do
11                  ./$1 $tam >> ./$1.dat
12                  i=$[$i+1]
13                  tam=$[$tam+$3]
14          done
15  else
16          echo "Error de argumentos"
17  fi
```

2.6.4. Script de *gnuplot*

```
1   #!/usr/bin/gnuplot
2
3   set xlabel "Tamanio del problema"
4   set ylabel "Tiempo (seg)"
5   set logscale
6   set terminal png size 640,480
7   set output 'busqueda_binaria.png'
8   plot 'busqueda_binaria.dat' with lines
9   set output 'busqueda_ternaria.png'
10  plot 'busqueda_ternaria.dat' with lines
```

2.6.5. Script automatizado

```
1   #!/bin/bash
2           echo "Compilando..."
3           g++ -o busqueda_binaria busqueda_binaria.cpp -O3 &&
4           g++ -o busqueda_ternaria busqueda_ternaria.cpp -O3 &&
5           rm -f ./busqueda_binaria.dat ;
6           rm -f ./busqueda_ternaria.dat ;
7           echo "Ejecutando búsqueda binaria..." ;
8           ./individual.sh busqueda_binaria 50000000 20000000;
```

```
9           echo "Ejecutando búsqueda ternaria..." ;
10          ./individual.sh busqueda_ternaria 50000000 20000000;
11          echo "Generando gráficas..." ;
12          ./gnuplot.sh ;
```