



ugr

Universidad
de Granada

ALGORÍTMICA
GRADO EN INGENIERÍA INFORMÁTICA

Ejercicio de clase

Búsqueda ternaria

Autores

Carlos Sánchez Páez



DECSAI

Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

CURSO 2017-2018

Índice

1. Enunciado	1
2. Resolución	1
2.1. Metodología	1
2.2. Resultados obtenidos	1
2.3. Cálculo de la constante oculta	1
2.4. Gráficas	1
2.5. Conclusiones	2
2.6. Anexo:Algoritmos desarrollados	3
2.6.1. Búsqueda binaria	3
2.6.2. Búsqueda ternaria	4
2.6.3. Script para múltiples ejecuciones	5
2.6.4. Script de <i>gnuplot</i>	5
2.6.5. Script automatizado	5

Índice de cuadros

1. Tamaños para la ejecución	1
2. Tiempos obtenidos (seg)	1
3. Bondad del ajuste	1

Índice de figuras

1. Eficiencia empírica. Búsqueda binaria	1
2. Eficiencia empírica. Búsqueda ternaria	2
3. Eficiencia híbrida. Búsqueda binaria	2
4. Eficiencia híbrida. Búsqueda ternaria	2

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).

Algoritmo	Tamaño inicial	Tamaño final	Incremento
Búsqueda Binaria	100.000.000	340.000.000	10.000.000
Búsqueda Ternaria			

Cuadro 1: Tamaños para la ejecución

2.2. Resultados obtenidos

Tamaño del problema	Búsqueda Binaria	Búsqueda Ternaria
---------------------	------------------	-------------------

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		
Búsqueda Ternaria		

Cuadro 3: Bondad del ajuste

2.4. Gráficas

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

2.6. Anexo:Algoritmos desarrollados

2.6.1. Búsqueda binaria

```
1  #define TEST 0          //Imprimir o no el resultado
2  #include <iostream>
3  #include <chrono>
4  #include <ctime>
5  #include <ratio>
6  #include <chrono>
7  using namespace std;
8  using namespace std::chrono;
9
10 int busquedaBinaria(const int *v, const int i, const int j, const int buscado) {
11     //Caso base: nos cruzamos
12     if (i > j) {
13         return -1;
14     }
15     else {
16         int mitad = (i + j) / 2;
17         if (v[mitad] == buscado)           //Acertamos
18             return mitad;
19         else if (buscado < v[mitad])       //Buscamos hacia la izquierda
20             return busquedaBinaria(v, i, mitad - 1, buscado);
21         else                               //Buscamos hacia la derecha
22             return busquedaBinaria(v, mitad + 1, j, buscado);
23     }
24 }
25 int main(int argc, char **argv) {
26     if (argc != 2) {
27         cerr << "Uso del programa: " << argv[0] << " <tamaño>" << endl;
28         exit(-1);
29     }
30     int tam = atoi(argv[1]);
31     int *v = new int[tam];
32     high_resolution_clock::time_point tantes;
33     high_resolution_clock::time_point tdespues;
34     duration<double> tiempo;
35
36     //Inicializar vector con valores aleatorios
37     srand (time(NULL));
38
39     for (int i = 0; i < tam; i++)
40         v[i] = rand() ;
41     // Peor caso: no está
42     tantes = high_resolution_clock::now();
43     int pos = busquedaBinaria(v, 0, tam, -1);
44     tdespues = high_resolution_clock::now();
45     tiempo = duration_cast<duration<double>>(tdespues - tantes);
46     #if TEST
47     cout << "Posición: " << pos << endl;
48     #endif
49     cout << tam << "\t\t" << tiempo.count() << endl;
50
51     delete []v;
52 }
```

2.6.2. Búsqueda ternaria

```
1  #define TEST 0           //Imprimir o no el resultado
2  #include <iostream>
3  #include <chrono>
4  #include <ctime>
5  #include <ratio>
6  #include <chrono>
7  using namespace std;
8  using namespace std::chrono;
9
10 int busquedaTernaria(const int *v, const int i, const int j, const int buscado) {
11     //Caso base: nos cruzamos
12     if (i > j)
13         return -1;
14     else {
15         int tam = j - i;
16         int tercio = i + (tam / 3);
17         int dostercio = i + (tam * 2) / 3;
18         if (v[tercio] == buscado) //Acertamos
19             return tercio;
20         else if (buscado < v[tercio]) //Primer tercio
21             busquedaTernaria(v, i, tercio - 1, buscado);
22         else {
23             if (v[dostercio] == buscado) //Acertamos
24                 return dostercio;
25             else if (buscado > v[dostercio]) //Tercer tercio
26                 busquedaTernaria(v, dostercio + 1, j, buscado);
27             else //Segundo tercio
28                 busquedaTernaria(v, tercio + 1, dostercio - 1, buscado);
29         }
30     }
31 }
32
33 int main(int argc, char **argv) {
34     if (argc != 2) {
35         cerr << "Uso del programa: " << argv[0] << " <tamaño>" << endl;
36         exit(-1);
37     }
38     int tam = atoi(argv[1]);
39     int *v = new int[tam];
40     high_resolution_clock::time_point tantes;
41     high_resolution_clock::time_point tdespues;
42     duration<double> tiempo;
43
44     //Inicializar vector con valores aleatorios
45     srand (time(NULL));
46
47     for (int i = 0; i < tam; i++)
48         v[i] = rand() ;
49     // Peor caso: no está
50     tantes = high_resolution_clock::now();
51     int pos = busquedaTernaria(v, 0, tam, -1);
52     tdespues = high_resolution_clock::now();
53     tiempo = duration_cast<duration<double>>(tdespues - tantes);
54
55     #if TEST
56     cout << "Posición: " << pos << endl;
```

```

57 #endif
58     cout << tam << "\t\t" << tiempo.count() << endl;
59
60     delete []v;
61 }

```

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 "Tamano del problema"
4  set ylabel "Tiempo (seg)"
5  set terminal png size 640,480
6  set output 'busqueda_binaria.png'
7  plot 'busqueda_binaria.dat' with lines
8  set output 'busqueda_ternaria.png'
9  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 &&
4      g++ -o busqueda_ternaria busqueda_ternaria.cpp &&
5      rm -f ./busqueda_binaria.dat ;
6      rm -f ./busqueda_ternaria.dat ;
7      echo "Ejecutando búsqueda binaria..." ;
8      ./individual.sh busqueda_binaria 10000000 10000000;
9      echo "Ejecutando búsqueda ternaria..." ;
10     ./individual.sh busqueda_ternaria 10000000 10000000;
11     echo "Generando gráficas..." ;
12     ./gnuplot.sh ;

```