Algorítmica

Grado en Ingeniería Informática

Ejercicio de clase

Divide y Vencerás

Autores
Carlos Sánchez Páez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2017-2018

Índice

Índice de cuadros

1. Enunciado

Implementar un algoritmo sencillo para calcular el valor máximo de un vector y otro basado en divide y vencerás. Realizar un estudio empírico e híbrido de su eficiencia.

2. Resolución

2.1. Algoritmo desarrollado

Para resolver el ejercicio he implementado el siguiente programa:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <climits>

using namespace std;

int hallarMaximo(const int *v, const int inicio, const int fin) {
        int max = v[inicio];
        for (int i = inicio + 1; i < fin; i++)
                if (v[i] > max)
                        max = v[i];
        return max;
}

int hallarMaximoDyV(const int *v,const int i,const int j){
        if(i==j){
                return v[i];
        }
        else{
                int mitad=(i+j)/2;
                int max_izquierda=hallarMaximoDyV(v,i,mitad);
                int max_derecha=hallarMaximoDyV(v,mitad+1,j);
                if(max_izquierda>max_derecha)
                        return max_izquierda;
                else
                        return max_derecha;
        }
```

```
29 }
30
31 int main(int argc, char **argv) {
32         if (argc != 2) {
33                 cerr << "Falta el tamaño del vector.";
34                 exit(-1);
35         }
36
37         int tam = atoi(argv[1]);
38         int *v = new int[tam];
39         int max;
40         clock_t tantes;
41         clock_t tdespues;
42
43         //Inicializar vector con valores aleatorios
44         srand (time(NULL));
45
46         for (int i = 0; i < tam; i++)
47                 v[i] = rand() ;
48
49         //Algoritmo sencillo
50
51         cout<<"\t\tTamaño "<<tam<<endl;
52         tantes = clock();
53         max = hallarMaximo(v, 0, tam);
54         tdespues = clock();
55         cout << "Algoritmo sencillo:\tMáximo: " << max
56         << "\tTiempo: " << ((double)(tdespues - tantes))
57          / CLOCKS_PER_SEC << endl;
58
59         //Algoritmo DyV
60
61         tantes = clock();
62         max = hallarMaximoDyV(v, 0,tam);
63         tdespues = clock();
64         cout << "Algoritmo DyV:\tMáximo:" << max
65         << "\tTiempo: " << ((double)(tdespues - tantes))
66           / CLOCKS_PER_SEC << endl;
67 }
```

2.2. Método básico

El método básico sigue el siguiente algoritmo:

1. Asigno el máximo al primer elemento del vector
2. Itero desde $v[1]$ hasta $v[tam]$
   - Si encuentro un elemento mayor que el máximo almacenado, lo sustituyo.
3. Devuelvo el máximo.

```
int hallarMaximo(const int *v, const int inicio, const int fin) {
        int max = v[inicio];
        for (int i = inicio + 1; i < fin; i++)
                if (v[i] > max)
                        max = v[i];
        return max;
}
        ...
        max = hallarMaximo(v, 0, tam);
```

Como vemos, su eficiencia *teórica* es $O(n)$.

2.3. Método Divide y Vencerás

En el método Divide y Vencerás hacemos lo siguiente:

1. Caso base: i=j. Devolvemos $v[i] = v[j]$
2. Lanzamos el algoritmo dos veces, desde i a la mitad y desde la mitad al final.
3. Devolvemos el máximo de entre las dos ejecuciones anteriores.

```
int hallarMaximoDyV(const int *v,const int i,const int j){
        if(i==j){
                return v[i];
        }
        else{
                int mitad=(i+j)/2;
                int max_izquierda=hallarMaximoDyV(v,i,mitad);
                int max_derecha=hallarMaximoDyV(v,mitad+1,j);
```

```
                if(max_izquierda>max_derecha)
                        return max_izquierda;
                else
                        return max_derecha;
        }
}
        ...
        max = hallarMaximoDyV(v, 0,tam);
```

Como vemos, su eficiencia es $O(\frac{n}{2}) = O(n)$

2.4. Estudio de eficiencia

2.4.1. Eficiencia empírica

Para realizar un estudio de la eficiencia empírica, ejecutaremos el programa 25 veces con tamaños ascendentes mediante un script:

```
#!/bin/bash
if [ $# -eq 3 ]
then
i="0"
tam=$2
#Primer argumento: programa a ejecutar
#Segundo argumento: tamaño inicial
#Tercer argumento : incremento
while [ $i -lt 25 ]
do
        ./$1 $tam >> ./$1.dat
        i=$[$i+1]
        tam=$[$tam+$3]
done
else
echo "Error de argumentos"
fi
```

Tamaño inicial	Tamaño final	Incremento
1.000.000	3.400.000	100.000

Cuadro 1: Parámetros de ejecución

Los tiempos obtenidos son los siguientes:

Tamaño del vector	**Algoritmo sencillo**	**Algoritmo DyV**
1.000.000	0.002843	0.01785
1.100.000	0.003078	0.019833
1.200.000	0.003348	0.021738
1.300.000	0.003672	0.023498
1.400.000	0.00399	0.025195
1.500.000	0.004231	0.026859
1.600.000	0.004494	0.02824
1.700.000	0.004768	0.030195
1.800.000	0.005065	0.031905
1.900.000	0.005365	0.03421
2.000.000	0.005717	0.035741
2.100.000	0.005959	0.036774
2.200.000	0.006238	0.039702
2.300.000	0.00655	0.041812
2.400.000	0.006852	0.043164
2.500.000	0.007097	0.04456
2.600.000	0.007376	0.047041
2.700.000	0.007694	0.048758
2.800.000	0.007935	0.050333
2.900.000	0.00823	0.051901
3.000.000	0.008527	0.053997
3.100.000	0.008772	0.055266
3.200.000	0.009132	0.056647
3.300.000	0.009391	0.05865
3.400.000	0.009629	0.060409

Cuadro 2: Tiempos de ejecución (s)

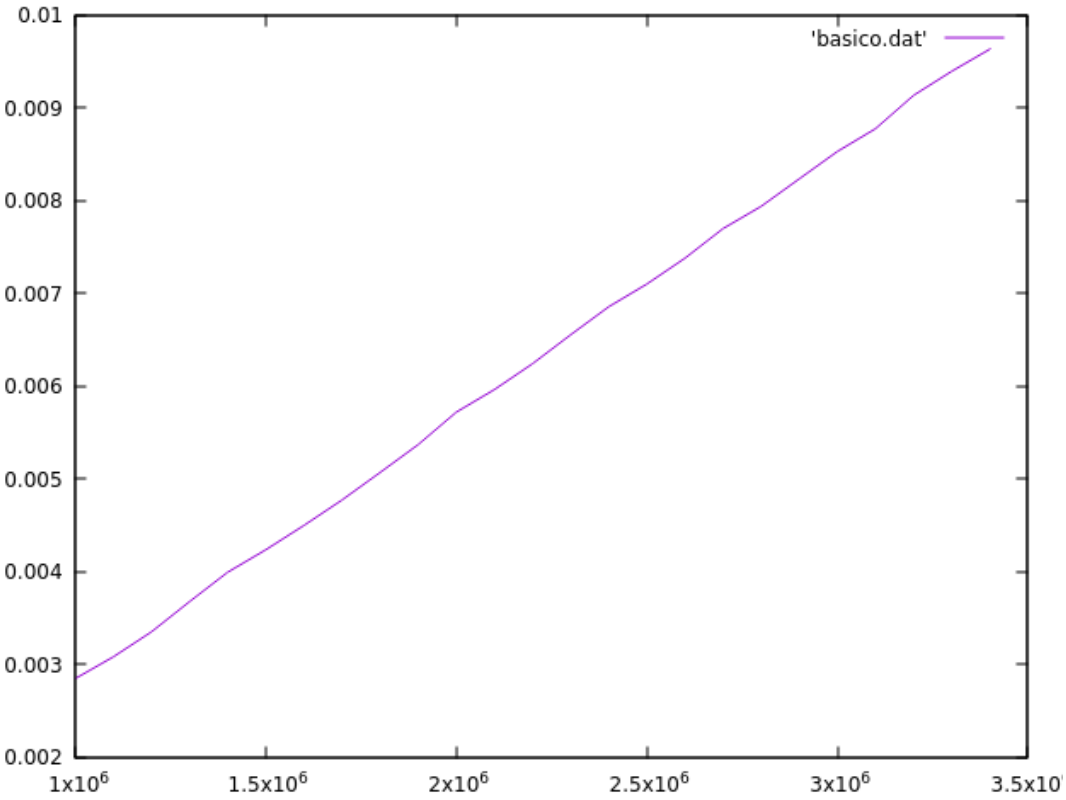

Figura 1: Eficiencia empírica. Algoritmo básico

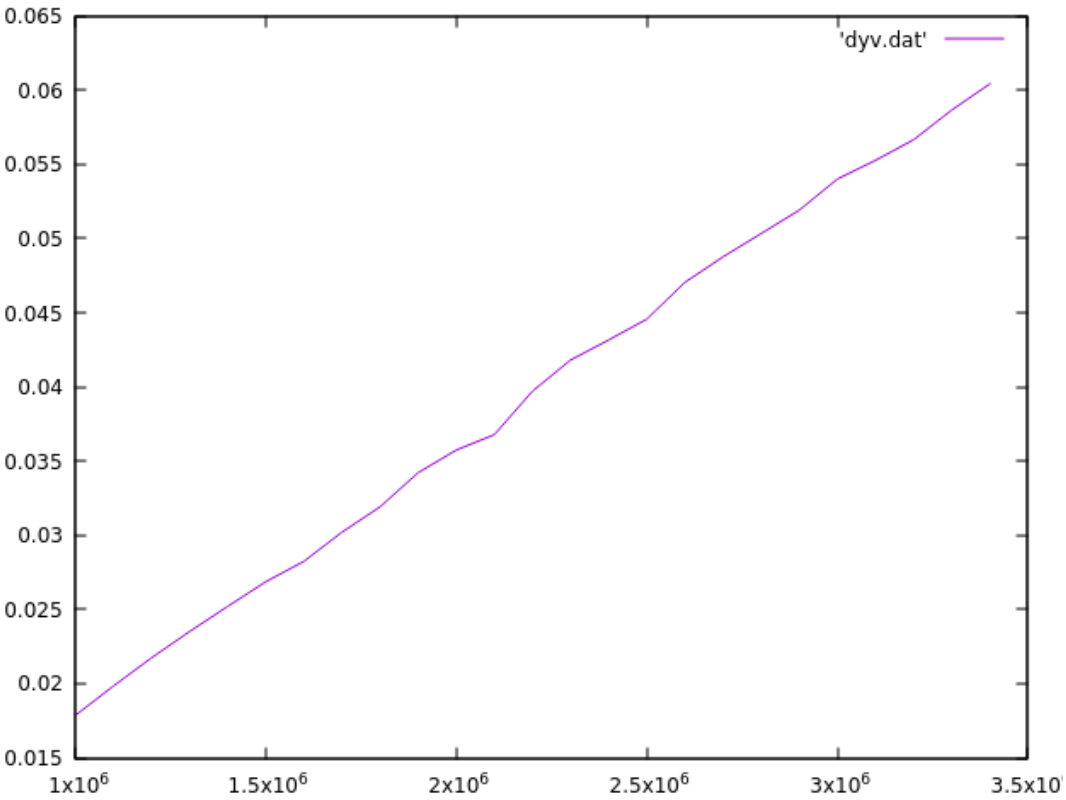

Figura 2: Eficiencia empírica. Algoritmo Divide y Vencerás

2.4.2. Eficiencia híbrida

Para esta sección, haremos una regresión mediante *gnuplot* de los datos empíricos obtenidos a la función $f(x) = a \cdot x$ para hallar la constante oculta.

Algoritmo	**Valor de la constante oculta**	**Porcentaje de error**
Básico	2.83765e-09	0.09622 %
Divide y Vencerás	1.78876e-08	0.1751 %

Cuadro 3: Bondad del ajuste

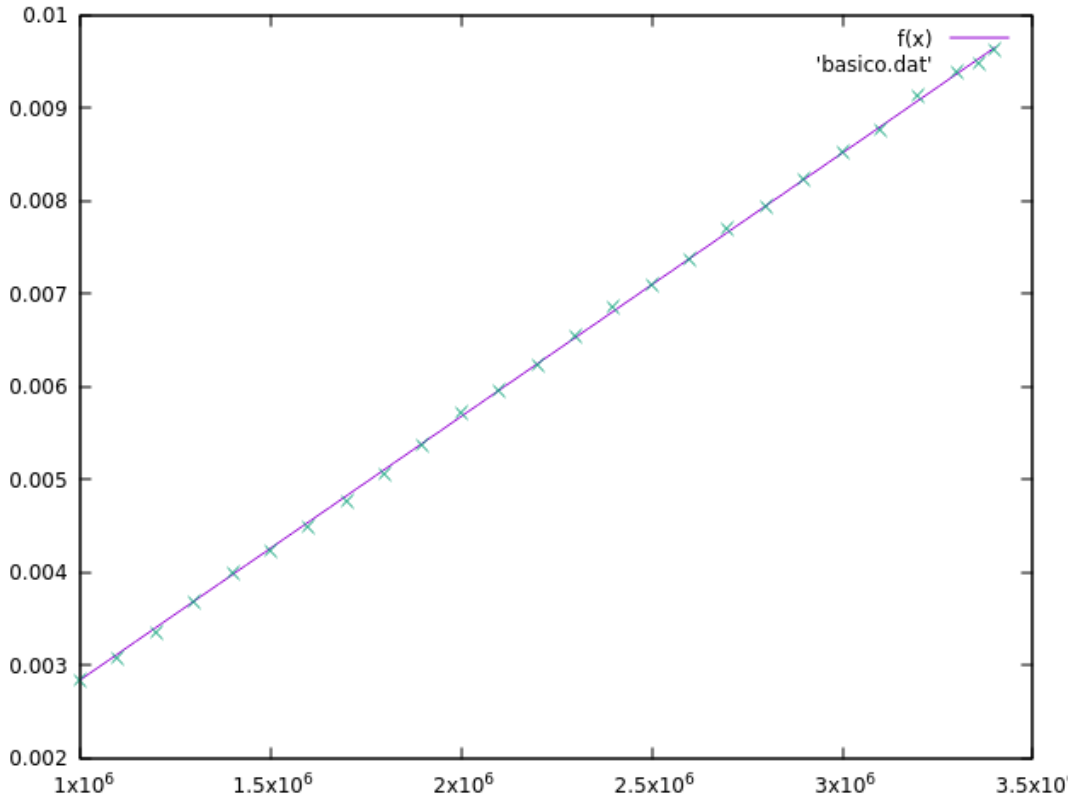

Figura 3: Eficiencia híbrida. Algoritmo básico

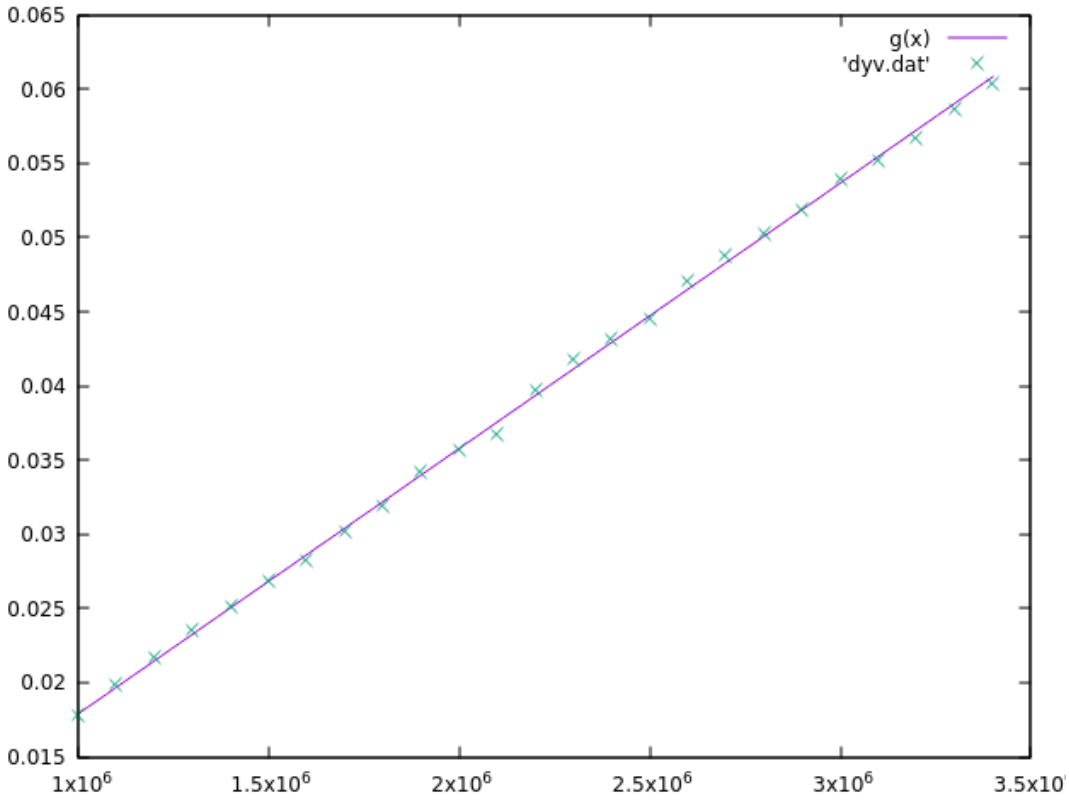

Figura 4: Eficiencia híbrida. Algoritmo Divide y Vencerás

2.5. Conclusiones

Como podemos ver, gracias al método *Divide y Vencerás* conseguimos rebajar la constante oculta del algoritmo inmediato. Sin embargo, sigue siendo mejor utilizar el algoritmo básico debido a los costes de la recursión que requiere Divide y Vencerás.