

Memoria de divide y vencerás

Rubén Morales Pérez Francisco Javier Morales Piqueras
Bruno Santindrian Manzanedo Ignacio de Loyola Barragan Lozano
Francisco Leopoldo Gallego Salido

11 de abril de 2016

Índice

1. Introducción	2
2. Mezclando k vectores ordenados	3
2.1. Estudio preliminar	3
2.2. Tiempo de ejecución	3
2.3. Mezcla con divide y vencerás	3
2.4. Estudio empírico e híbrido	3
3. Comparación de preferencias (opcional)	4
3.1. Automatizando el problema	4
3.2. Fuerza bruta	5
3.3. Divide y vencerás	7
3.4. Comparación	11
4. Bibliografía	12

1. Introducción

Divide y vencerás es una técnica algorítmica que consiste en resolver un problema dividiéndolo en problemas más pequeños y combinando las soluciones. El proceso de división continúa hasta que los subproblemas llegan a ser lo suficientemente sencillos como para una resolución directa.

El hecho de que el tamaño de los subproblemas sea estrictamente menor que el tamaño original del problema nos garantiza la convergencia hacia los casos elementales. El coste computacional se determina resolviendo relaciones de recurrencia.

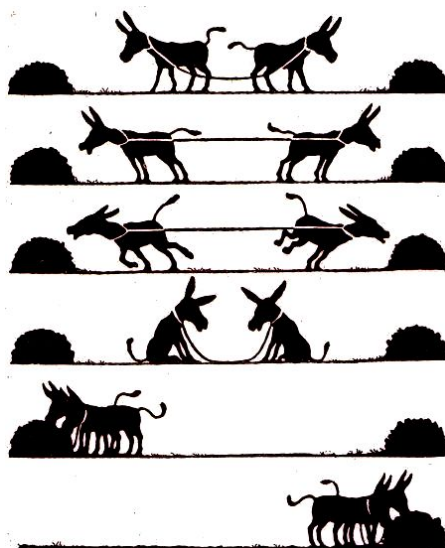


Figura 1: Subproblemas

Una forma de dirigir las cartas siguiendo este método (creada por Donald Knuth, autor de The Art of Computer Programming) es separarlas en bolsas diferentes en función de su área geográfica y cada una a su vez se ordena en lotes para subregiones más pequeñas.

Un ejemplo anterior a Cristo donde se usa “divide y vencerás” con un único subproblema es el algoritmo de Euclides para computar el máximo común divisor de dos números.

$$\begin{array}{rcl} 11312 & = & 500 \cdot 22 + 312, \dots[1] \\ 500 & = & 312 \cdot 1 + 188, \dots[2] \\ 312 & = & 188 \cdot 1 + 124, \dots[3] \\ 188 & = & 124 \cdot 1 + 64, \dots[4] \\ 124 & = & 64 \cdot 1 + 60, \dots[5] \\ 64 & = & 60 \cdot 1 + 4 \dots[6] \end{array}$$

Figura 2: Euclides

2. Mezclando k vectores ordenados

Se tienen k vectores ordenados (de menor a mayor), cada uno con n elementos, y queremos combinarlos en un único vector ordenado (con kn elementos). Una posible alternativa consiste en, utilizando un algoritmo clásico, mezclar los dos primeros vectores, posteriormente mezclar el resultado con el tercero, y así sucesivamente.

- ¿Cuál sería el tiempo de ejecución de este algoritmo?
- Diseñe, analice la eficiencia e implemente un algoritmo de mezcla más eficiente, basado en *divide y vencerás*:
- Realizar también un estudio empírico e híbrido de la eficiencia de ambos algoritmos.

2.1. Estudio preliminar

Planteando el problema es posible imponer una cota superior teórica a la mezcla. Teniendo en cuenta que hay kn elementos, si aplicásemos un algoritmo con eficiencia $O(n) = n \log(n)$ deducimos que podemos encontrar un algoritmo de ordenación básica con eficiencia $O(k, n) = nk \log(nk)$. En este caso estaríamos representando los k vectores de n elementos como un único vector, sin aprovechar aún el hecho de que parte del vector está ordenado.

2.2. Tiempo de ejecución

2.3. Mezcla con *divide y vencerás*

2.4. Estudio empírico e híbrido

3. Comparación de preferencias (opcional)

Muchos sitios web intentan comparar las preferencias de dos usuarios para realizar sugerencias a partir de las preferencias de usuarios con gustos similares a los nuestros. Dado un ranking de n productos (p.ej. películas) mediante el cual los usuarios indicamos nuestras preferencias, un algoritmo puede medir la similitud de nuestras preferencias contando el número de inversiones: dos productos i y j están invertidos en las preferencias de A y B si el usuario A prefiere el producto i antes que el j , mientras que el usuario B prefiere el producto j antes que el i . Esto es, cuantas menos inversiones existan entre dos rankings, más similares serán las preferencias de los usuarios representados por esos rankings.

Por simplicidad podemos suponer que los productos se pueden identificar mediante enteros $1, \dots, n$, y que uno de los rankings siempre es $1, \dots, n$ (si no fuese así bastaría reenumerarlos) y el otro es a_1, a_2, \dots, a_n , de forma que dos productos i y j están invertidos si $i < j$ pero $a_i > a_j$. De esta forma nuestra representación del problema será un vector de enteros v de tamaño n , de forma que $v[i] = a_i / i = 1, \dots, n$.

El objetivo es diseñar, analizar la eficiencia e implementar un algoritmo divide y vencerás para medir la similitud entre dos rankings. Compararlo con el algoritmo de fuerza bruta obvio. Realizar también un estudio empírico e híbrido de la eficiencia de ambos algoritmos.

3.1. Automatizando el problema

A la hora de generar los ejecutables, los datos y las gráficas hemos usado los siguientes scripts:

script.sh

```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]
4  then
5      echo "Uso: $0 <nombre>"
6      exit 1
7  fi
8
9  mkdir ../Graficas 2> /dev/null
10 mkdir ../Datos 2> /dev/null
11
12 # Fuerza bruta
13 g++ -std=c++11 -D GP_OUT ../src/fuerza_bruta.cpp
14 nelementos=10
15 echo "" > datos.dat
16 while [ $nelementos -lt 5000 ]; do
17     ./a.out $nelementos >> datos.dat
18     let nelementos=nelementos+10
19 done
20 gnuplot ../gnuplot/fuerza_bruta.gp
21
22 mv grafica.png ../Graficas/fuerza_bruta_$1.png
23 mv datos.dat ../Datos/fuerza_bruta_$1.dat
24 mv fit.log ../Datos/fit_fuerza_bruta_$1.log
25 echo "Fuerza bruta completado"
26
```

```

27 # DyV
28 g++ -std=c++11 -D GP_OUT ../src/dyv.cpp
29 nelementos=10
30 echo "" > datos.dat
31 while [ $nelementos -lt 5000 ]; do
32     ./a.out $nelementos >> datos.dat
33     let nelementos=nelementos+10
34 done
35 gnuplot ./gnuplot/dyv.gp
36
37 mv grafica.png ../Graficas/dyv_$1.png
38 mv datos.dat ../Datos/dyv_$1.dat
39 mv fit.log ../Datos/fit_dyv_$1.log
40 echo "DyV completado"

```

Para conseguir los datos y gráficas hemos usado:

fuerza_bruta.gp

```

1 set terminal pngcairo
2 set output "grafica.png"
3
4 set title "Fuerza bruta"
5 set xlabel "Tamano del vector"
6 set ylabel "Tiempo (s)"
7 set fit quiet
8 f(x) = a*x*x + b*x + c
9 fit f(x) "datos.dat" via a, b, c
10 plot "datos.dat", f(x)

```

dyv.gp

```

1 set terminal pngcairo
2 set output "grafica.png"
3
4 set title "Divide y vencer\'as"
5 set xlabel "Tamano del vector"
6 set ylabel "Tiempo (s)"
7 set fit quiet
8 f(x) = a*x*x + b*x + c
9 fit f(x) "datos.dat" via a, b, c
10 plot "datos.dat", f(x)

```

3.2. Fuerza bruta

El programa usado para calcular los tiempos del algoritmo de fuerza bruta es:

```

1 #include <iostream>
2 #include <vector>

```

```

3  #include <algorithm>
4  #include <ctime>
5  #include <cstdlib>
6  #include <chrono>
7  using namespace std;
8
9
10 int inversiones(vector<int> v){
11     int count = 0;
12     int size = v.size();
13     for (int i=0; i < size; i++)
14         for (int j=i+1; j < size; j++)
15             if (v[i] > v[j])
16                 count++;
17
18     return count;
19 }
20
21
22 int main(int argc, char** argv){
23     if (argc < 2){
24         cerr << "Formato " << argv[0] << " num_elem" << endl;
25         return -1;
26     }
27
28     int n = atoi(argv[1]);
29     vector<int> rankings(n);
30     srand(time(0));
31     for (int i=0; i < n; i++)
32         rankings[i] = i;
33
34     random_shuffle(rankings.begin(), rankings.end());
35
36     chrono::high_resolution_clock::time_point tantes, tdespues;
37     chrono::duration<double> transcurrido;
38     int n_inv;
39
40     tantes = chrono::high_resolution_clock::now();
41     n_inv = inversiones(rankings);
42     tdespues = chrono::high_resolution_clock::now();
43
44     transcurrido = chrono::duration_cast<chrono::duration<double>>(tdespues -
45         tantes);
46
47     #ifndef GP_OUT
48         cout << "ranking: ";
49         for (int i=0; i<n; i++)
50             cout << rankings[i] << " ";
51         cout << endl;
52         cout << "Num inversiones: " << n_inv << endl;
53     #else
54         cout << n << " " << transcurrido.count() << endl;
55     #endif

```

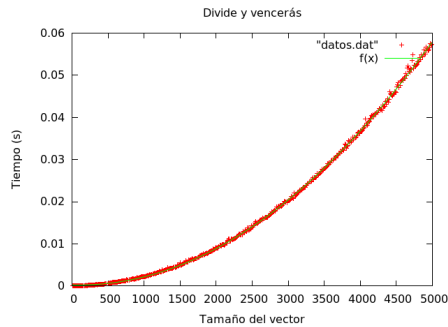


Figura 3: Subproblemas

3.3. Divide y vencerás

El mismo problema usando un programa que utiliza divide y vencerás:

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <ctime>
5  #include <climits>
6  #include <cstdlib>
7  #include <chrono>
8  using namespace std;
9
10
11
12  /* ***** */
13  /* M\'etodo de ordenaci\'on por mezcla */
14
15  /**
16   @brief Devuelve el numero de inversiones de un ranking.
17
18   @param T: vector de elementos. Debe tener num_elem elementos.
19             Es MODIFICADO.
20   @param num_elem: n\'umero de elementos. num_elem > 0.
21
22   @return el numero de inversiones
23
24   Calcula el numero de inversiones aplicando el algoritmo de mezcla.
25   Como consecuencia tambien ordena el vector T.
26  */
27  inline static
28  int inversiones(int T[], int num_elem);
29
30
31
32  /**

```

```

33     @brief Devuelve el n\úmero de inversiones de un ranking.
34
35     @param T: vector de elementos. Tiene un n\úmero de elementos
36               mayor o igual a final. Es MODIFICADO.
37     @param inicial: Posici\ón que marca el incio de la parte del
38               vector a ordenar.
39     @param final: Posici\ón detr\as de la \última de la parte del
40               vector a ordenar.
41               inicial < final.
42
43     Calcula el n\úmero de inversiones aplicando el algoritmo de mezcla.
44     Como consecuencia también ordena el vector T.
45 */
46 void inversiones_recursivo(int T[], int inicial, int final, int&
    num_inversiones);
47
48
49 /**
50     @brief Ordena un vector por el m\étodo de inserci\ón.
51
52     @param T: vector de elementos. Debe tener num_elem elementos.
53               Es MODIFICADO.
54     @param num_elem: n\úmero de elementos. num_elem > 0.
55
56     Cambia el orden de los elementos de T de forma que los dispone
57     en sentido creciente de menor a mayor.
58     Aplica el algoritmo de inserci\ón.
59 */
60 inline static
61 void insercion(int T[], int num_elem);
62
63
64 /**
65     @brief Ordena parte de un vector por el m\étodo de inserci\ón.
66
67     @param T: vector de elementos. Tiene un n\úmero de elementos
68               mayor o igual a final. Es MODIFICADO.
69     @param inicial: Posici\ón que marca el incio de la parte del
70               vector a ordenar.
71     @param final: Posici\ón detr\as de la \última de la parte del
72               vector a ordenar.
73               inicial < final.
74
75     Cambia el orden de los elementos de T entre las posiciones
76     inicial y final - 1 de forma que los dispone en sentido creciente
77     de menor a mayor.
78     Aplica el algoritmo de la inserci\ón.
79 */
80 static void insercion_lims(int T[], int inicial, int final);
81
82
83 /**
84     @brief Mezcla dos vectores ordenados sobre otro.
85

```



```

86  @param T: vector de elementos. Tiene un n\úmero de elementos
87           mayor o igual a final. Es MODIFICADO.
88  @param inicial: Posici\ón que marca el incio de la parte del
89           vector a escribir.
90  @param final: Posici\ón detr\as de la \última de la parte del
91           vector a escribir
92           inicial < final.
93  @param U: Vector con los elementos ordenados.
94  @param V: Vector con los elementos ordenados.
95           El n\úmero de elementos de U y V sumados debe coincidir
96           con final - inicial.
97
98  @return: El n\úmero de elementos que estaban desordenados
99
100  En los elementos de T entre las posiciones inicial y final - 1
101  pone ordenados en sentido creciente, de menor a mayor, los
102  elementos de los vectores U y V.
103  */
104  int fusion(int T[], int inicial, int final, int U[], int V[]);
105
106
107
108  /**
109   Implementaci\ón de las funciones
110  */
111
112
113  int inversiones(int T[], int num_elem)
114  {
115      int n = 0;
116      inversiones_recursivo(T, 0, num_elem, n);
117      return n;
118  }
119
120  void inversiones_recursivo(int T[], int inicial, int final, int&
      num_inversiones)
121  {
122      if (final - inicial < 2)
123      {
124          return;
125      } else {
126          int k = (final - inicial)/2;
127
128          int * U = new int [k - inicial];
129          int l, l2;
130          for (l = 0, l2 = inicial; l < k; l++, l2++)
131              U[l] = T[l2];
132
133          int * V = new int [final - k];
134          for (l = 0, l2 = k; l < final - k; l++, l2++)
135              V[l] = T[l2];
136
137          inversiones_recursivo(U, 0, k, num_inversiones);
138          inversiones_recursivo(V, 0, final - k, num_inversiones);

```

```

139         num_inversiones += fusion(T, inicial, final, U, V);
140         delete [] U;
141         delete [] V;
142     };
143 }
144
145
146 int fusion(int T[], int inicial, int final, int U[], int V[])
147 {
148     int n_desordenados = 0;
149     int k = (final-inicial)/2;
150     for (int i =0; i < k-inicial; i++)
151         for (int j=0; j < final-k; j++)
152             if (U[i] > V[j])
153                 n_desordenados++;
154
155     int i, j;
156     for (i =0; i < k-inicial; i++)
157         T[i] = U[i];
158     for (i=0, j=k-inicial; j < final-k; i++, j++)
159         T[j] = V[i];
160
161     return n_desordenados;
162 }
163
164
165 int main(int argc, char** argv){
166     if (argc < 2){
167         cerr << "Formato " << argv[0] << " num_elem" << endl;
168         return -1;
169     }
170
171     int n = atoi(argv[1]);
172     int* rankings = new int[n];
173     srand(time(0));
174     for (int i=0; i < n; i++){
175         rankings[i] = i;
176     }
177     random_shuffle(rankings, rankings+n);
178
179     #ifndef GP_OUT
180         cout << "ranking: ";
181         for (int i=0; i<n; i++)
182             cout << rankings[i] << " ";
183         cout << endl;
184     #endif
185
186     chrono::high_resolution_clock::time_point tantes, tdespues;
187     chrono::duration<double> transcurrido;
188     int n_inv;
189
190     tantes = chrono::high_resolution_clock::now();
191     n_inv = inversiones(rankings, n);
192     tdespues = chrono::high_resolution_clock::now();

```

```

193     transcurrido = chrono::duration_cast<chrono::duration<double>>(tdespues -
      tantes);
194
195 #ifndef GP_OUT
196     cout << "Num inversiones: " << n_inv << endl;
197 #else
198     cout << n << " " << transcurrido.count() << endl;
199 #endif
200 }

```

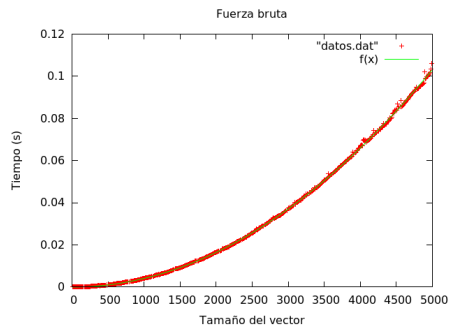


Figura 4: Subproblemas

3.4. Comparación

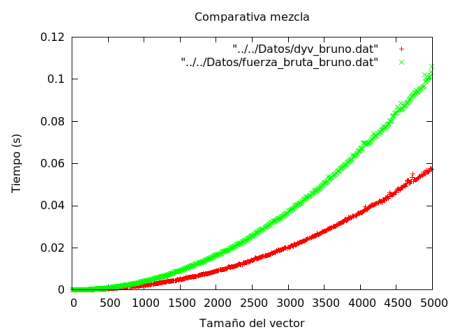


Figura 5: Subproblemas

4. Bibliografia