Algorítmica

grado en ingeniería informática

Ejercicio de clase

Umbral de *mergesort*

Autores
Carlos Sánchez Páez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2017-2018

Índice

Índice de cuadros

1. Enunciado

Realizar un estudio empírico para determinar un buen umbral para el algoritmo de ordenación por mezcla o mergesort (que utiliza como algoritmo auxiliar el algoritmo de ordenación por inserción para casos de tamaño menor que el umbral).

2. Resolución

2.1. Metodología

Para realizar el ejercicio, modificamos el código de *mergesort.cpp* disponible en *DECSAI* para que acepte el tamaño del umbral como parámetro de ejecución del programa.
La metodología será realizar 20 ejecuciones con un tamaño de vector constante (10.000.000) y un umbral variable para determinar el tamaño óptimo del límite.
Para elo, he desarrollado el siguiente *script*:

2.2. Script desarrollado

```
#!/bin/bash
if [ $# -eq 3 ]
then
        i="0"
        tam=$2
        #Primer argumento: programa a ejecutar
        #Segundo argumento: tamaño inicial del umbral
        #Tercer argumento : incremento
        while [ $i -lt 20 ]
        do
                ./$1 $tam 1000000 >> ./$1.dat
                i=$[$i+1]
                tam=$[$tam+$3]
        done
else
        echo "Error de argumentos"
fi
```

2.3. Datos de las ejecuciones

Comenzamos ejecutando desde 10 hasta 100:

Tamaño de umbral	Tiempo de ejecución (s)
10	0.246989
20	0.242702
30	0.245441
40	0.249388
50	0.25211
60	0.251404
70	0.283634
80	0.292154
90	0.283183
100	0.281779
110	0.283369
120	0.28545
130	0.362056
140	0.36134
150	0.361897
160	0.362099
170	0.362484
180	0.362892
190	0.362747
200	0.36144

Cuadro 1: Primera aproximación

Vemos que entre $t_{umbral} = 120$ y $t_{umbral} = 130$ se produce un gran salto. Por tanto, volvemos a lanzar el script para afinar a las unidades:

Tamaño de umbral	Tiempo de ejecución (s)
120	0.283637
121	0.285872
122	0.284597
123	0.360809
124	0.362452
125	0.360889
126	0.362977
127	0.361814
128	0.362678
129	0.361538
130	0.370429

Cuadro 2: Segunda aproximación

2.4. Conclusión

Por tanto, podemos concluir que un umbral adecuado para el algoritmo *mergesort* es **122**.

3. Anexo: Código fuente utilizado

```
#include <iostream>
using namespace std;
#include <ctime>
#include <cstdlib>
#include <climits>
#include <cassert>

/**
   @brief Ordena un vector por el método de mezcla.

   @param T: vector de elementos. Debe tener num_elem elementos.
             Es MODIFICADO.
   @param num_elem: número de elementos. num_elem > 0.

   Cambia el orden de los elementos de T de forma que los dispone
   en sentido creciente de menor a mayor.
   Aplica el algoritmo de mezcla.
*/
inline static
void mergesort(int T[], int num_elem, const int UMBRAL_MS);

/**
   @brief Ordena parte de un vector por el método de mezcla.

   @param T: vector de elementos. Tiene un número de elementos
                  mayor o igual a final. Es MODIFICADO.
   @param inicial: Posición que marca el incio de la parte del
                  vector a ordenar.
   @param final: Posición detrás de la última de la parte del
                  vector a ordenar.
                  inicial < final.

```

```
33      Cambia el orden de los elementos de T entre las posiciones
34      inicial y final - 1 de forma que los dispone en sentido
35      creciente de menor a mayor.
36      Aplica el algoritmo de la mezcla.
37  */
38  static void mergesort_lims(int T[], int inicial, int final,
39  const int UMBRAL_MS);
40
41  /**
42      @brief Ordena un vector por el método de inserción.
43
44      @param T: vector de elementos. Debe tener num_elem elementos.
45                Es MODIFICADO.
46      @param num_elem: número de elementos. num_elem > 0.
47
48      Cambia el orden de los elementos de T de forma que los dispone
49      en sentido creciente de menor a mayor.
50      Aplica el algoritmo de inserción.
51  */
52  inline static
53  void insercion(int T[], int num_elem);
54
55  /**
56      @brief Ordena parte de un vector por el método de inserción.
57
58      @param T: vector de elementos. Tiene un número de elementos
59                      mayor o igual a final. Es MODIFICADO.
60      @param inicial: Posición que marca el incio de la parte del
61                      vector a ordenar.
62      @param final: Posición detrás de la última de la parte del
63                      vector a ordenar.
64                      inicial < final.
65
66      Cambia el orden de los elementos de T entre las posiciones
67      inicial y final - 1 de forma que los dispone en sentido
68      creciente de menor a mayor.
69      Aplica el algoritmo de la inserción.
70  */
71  static void insercion_lims(int T[], int inicial, int final);
```

```
72
73  /**
74     @brief Mezcla dos vectores ordenados sobre otro.
75
76     @param T: vector de elementos. Tiene un número de elementos
77                     mayor o igual a final. Es MODIFICADO.
78     @param inicial: Posición que marca el incio de la parte del
79                     vector a escribir.
80     @param final: Posición detrás de la última de la parte del
81                     vector a escribir
82                     inicial < final.
83     @param U: Vector con los elementos ordenados.
84     @param V: Vector con los elementos ordenados.
85               El número de elementos de U y V sumados debe
86               coincidir con final - inicial.
87
88     En los elementos de T entre las posiciones inicial y final
89     - 1 pone ordenados en sentido creciente, de menor a mayor,
90     los elementos de los vectores U y V.
91  */
92  static void fusion(int T[], int inicial, int final,
93  int U[], int V[]);
94
95  /**
96     Implementación de las funciones
97  **/
98
99  inline static void insercion(int T[], int num_elem){
100         insercion_lims(T, 0, num_elem);
101 }
102
103
104 static void insercion_lims(int T[], int inicial, int final){
105         int i, j;
106         int aux;
107         for (i = inicial + 1; i < final; i++) {
108                 j = i;
109                 while ((T[j] < T[j - 1]) && (j > 0)) {
110                         aux = T[j];
```

```
111                         T[j] = T[j - 1];
112                         T[j - 1] = aux;
113                         j--;
114                 };
115         };
116 }
117
118
119 void mergesort(int T[], int num_elem, const int UMBRAL_MS){
120         mergesort_lims(T, 0, num_elem, UMBRAL_MS);
121 }
122
123 static void mergesort_lims(int T[], int inicial,
124 int final, const int UMBRAL_MS{
125         if (final - inicial < UMBRAL_MS)
126                 insercion_lims(T, inicial, final);
127         else {
128                 int k = (final - inicial) / 2;
129
130                 int * U = new int [k - inicial + 1];
131                 assert(U);
132                 int l, l2;
133                 for (l = 0, l2 = inicial; l < k; l++, l2++)
134                         U[l] = T[l2];
135                 U[l] = INT_MAX;
136
137                 int * V = new int [final - k + 1];
138                 assert(V);
139                 for (l = 0, l2 = k; l < final - k; l++, l2++)
140                         V[l] = T[l2];
141                 V[l] = INT_MAX;
142
143                 mergesort_lims(U, 0, k, UMBRAL_MS);
144                 mergesort_lims(V, 0, final - k, UMBRAL_MS);
145                 fusion(T, inicial, final, U, V);
146                 delete [] U;
147                 delete [] V;
148         };
149 }
```

```
150
151 static void fusion(int T[], int inicial, int final, int U[],
152  int V[]){
153         int j = 0;
154         int k = 0;
155         for (int i = inicial; i < final; i++){
156                 if (U[j] < V[k]) {
157                         T[i] = U[j];
158                         j++;
159                 } else {
160                         T[i] = V[k];
161                         k++;
162                 };
163         };
164 }
165
166 int main(int argc, char * argv[])
167 {
168
169         if (argc != 3){
170                 cerr << "Formato " << argv[0] << " <umbral>
171                 <num_elem>" << endl;
172                 return -1;
173         }
174         const int UMBRAL_MS = atoi(argv[1]);
175         int n = atoi(argv[2]);
176
177         int * T = new int[n];
178         assert(T);
179
180         srandom(time(0));
181
182         for (int i = 0; i < n; i++)
183                 T[i] = random();
184
185         clock_t tantes;
186         clock_t tdespues;
187         tantes = clock();
188         mergesort(T, n, UMBRAL_MS);
```

```
189             tdespues = clock();
190             cout << UMBRAL_MS << "\t" <<
191             ((double)(tdespues - tantes))
192              / CLOCKS_PER_SEC << endl;
193
194             delete [] T;
195
196             return 0;
197     };
```