Algorítmica: práctica 2

Mezclando k vectores ordenados

Grupo 2

Sofía Almeida Bruno Antonio Coín Castro María Victoria Granados Pozo
Miguel Lentisco Ballesteros José María Martín Luque

6 de abril de 2017

Introducción

El objetivo de esta práctica es diseñar un algoritmo *divide y vencerás* que se encargue de combinar k vectores ordenados. Además tenemos que analizar su eficiencia, implementarlo y compararlo con un algoritmo clásico.

Algoritmo clásico

A continuación se proporciona el código de la función `mezcla_vectores`, que utiliza un algoritmo clásico para mezclar k vectores en uno solo. El código del programa completo se puede encontrar en la carpeta *src*.

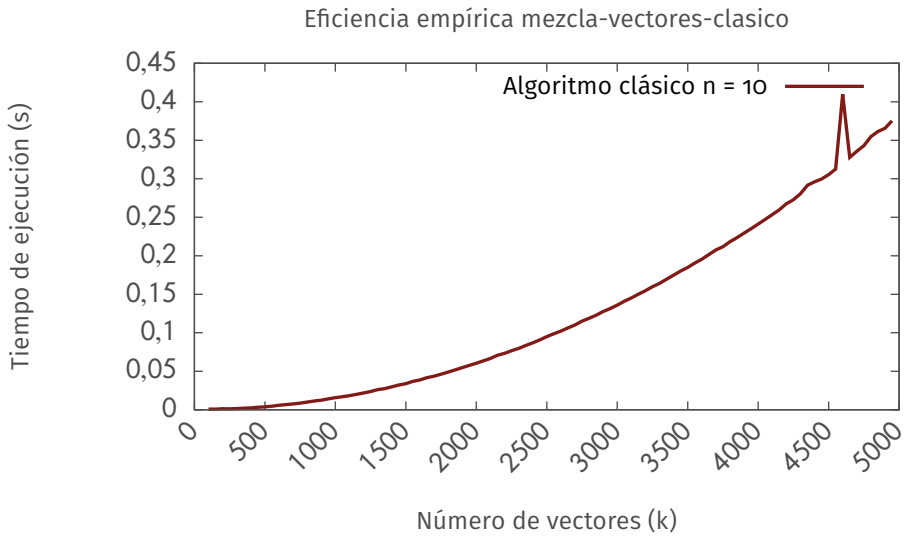
```
1  void merge(int T1[], int T2[], int S[], int n1, int n2) {
2    int p1 = 0, p2 = 0, p3 = 0;
3
4    while (p1 < n1 && p2 < n2) {
5      if (T1[p1] <= T2[p2]) {
6        S[p3] = T1[p1];
7        p1++;
8      }
9      else {
10       S[p3] = T2[p2];
11       p2++;
12     }
13
14     p3++;
15   }
16
17   while (p1 < n1) {
18     S[p3++] = T1[p1++];
19   }
20
21   while (p2 < n2) {
22     S[p3++] = T2[p2++];
23   }
24 }
25 int* mezcla_vectores(int** T, int k, int n) {
26   int* S = new int[k*n];  // Vector mezcla
27   assert(S);
28
29   if (k > 1) {
30     int* aux = new int [k*n];
31     assert(aux);
32
33     // Primera mezcla
34     merge(T[0], T[1], S, n, n);
35
36     // Resto de mezclas
37     for (int i = 2; i < k; i++) {
38       merge(S, T[i], aux, i*n, n);
39       swap(S, aux);   // Intercambiamos punteros
40     }
41
42     delete [] aux;
```

```
  }

  else {
    for (int i = 0; i < n; i++) {
      S[i] = T[0][i];
    }
  }

  return S;
}
```

Eficiencia teórica

Eficiencia empírica

En el gráfico que se muestra a continuación se muestran los resultados de la ejecución del algoritmo *clásico* con vectores de 10 elementos.



Eficiencia *híbrida*

Algoritmo *divide y vencerás* con vectores dinámicos

A continuación se proporciona el código de la función `mezclaDV`, que utiliza un algoritmo divide y vencerás (con vectores dinámicos) para mezclar k vectores en uno solo. El código del programa completo se puede encontrar en la carpeta *src*.

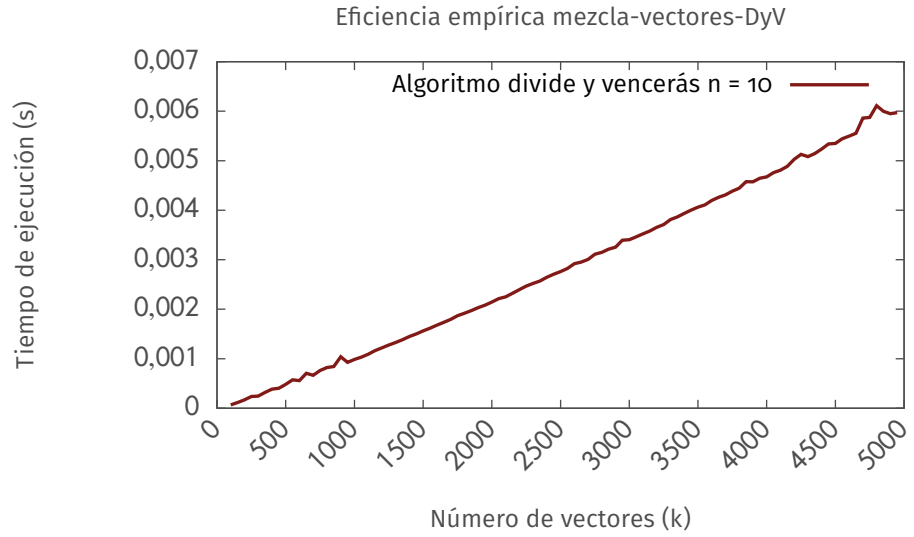
```
int* mezclaDV(int** T, int n, int start, int end) {
  int k = end - start + 1;  // Número de vectores

  // Caso base
  if (k == 1) {
    return T[start];
  }
```

```
8
9     // Caso general
10    else {
11      int middle = (start + end) / 2;
12      int n1 = middle - start + 1;
13      int n2 = end - (middle + 1) + 1;
14
15      // Divide
16      int* izqda = mezclaDV(T, n, start, middle);
17      int* dcha = mezclaDV(T, n, middle + 1, end);
18
19      // Vencerás
20      return merge(izqda, dcha, n * n1, n * n2);
21    }
22  }
```

Eficiencia teórica

Eficiencia empírica

En el gráfico que se muestra a continuación se muestran los resultados de la ejecución del algoritmo *divide y vencerás* con vectores dinámicos de 10 elementos.



Eficiencia *híbrida*

Algoritmo *divide y vencerás* con vectores de la STL

A continuación se proporciona el código de la función `mezclaDV`, que utiliza un algoritmo divide y vencerás (con vectores de la STL) para mezclar k vectores en uno solo. El código del programa completo se puede encontrar en la carpeta *src*.

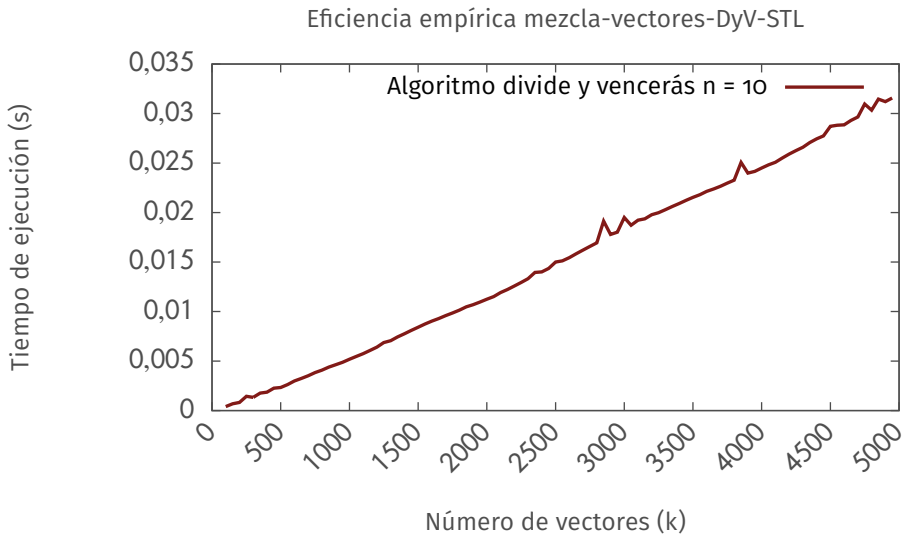
```
1   vector<int> mezclaDV(vector<vector<int>> vectores) {
```

```
2
3       // Casos base
4       if (vectores.size() < 1) {
5           vector<int> sol;
6           return sol;
7       } else if (vectores.size() == 1) {
8           return vectores[0];
9       } else if (vectores.size() == 2) {
10          return merge(vectores[0], vectores[1]);
11      }
12
13      vector<vector<int>>::iterator half = vectores.begin() + vectores.size() / 2;
14      vector<vector<int>> firstHalf(vectores.begin(), half), secondHalf(half + 1, vectores.end());
15
16      // Divide
17      vector<int> s1 = mezclaDV(firstHalf);
18      vector<int> s2 = mezclaDV(secondHalf);
19
20      // Vencerás
21      return merge(s1, s2);
22  }
```

Eficiencia teórica

Eficiencia empírica

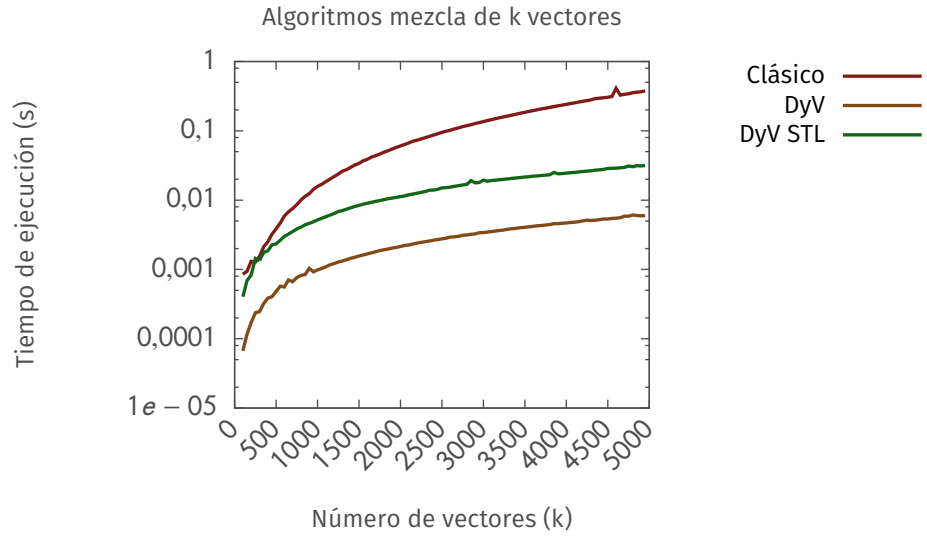
En el gráfico que se muestra a continuación se muestran los resultados de la ejecución del algoritmo *divide y vencerás* con vectores `std::vector` de 10 elementos.



Eficiencia *híbrida*

Comparación de la eficiencia

En el siguiente gráfico se puede observar de forma visual qué algoritmo es más eficiente. Como es de esperar, el algoritmo clásico es el más lento de todos. Algo más curioso quizás es que el algoritmo que utiliza vectores *dinámicos* es más rápido que el que usa la clase `vector` de la STL.



Anexo

Características de los ordenadores donde se ha ejecutado

1. Apple MacBook Pro, Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz, 8GB RAM.
   Compilador: clang-800.0.38
   Sistema operativo: macOS Sierra

2. Dell XPS 13, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 8GB RAM.
   Compilador: g++ 6.3.1
   Sistema operativo: Arch Linux