

Algorítmica: práctica 2

Mezclando k vectores ordenados

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

Grupo 2

4 de abril de 2017

Objetivo

Estudiar la eficiencia teórica, empírica e híbrida de 8 algoritmos. También, realizar comparaciones entre sus tiempos de ejecución, así como la influencia de otros factores, como S.O. utilizado o prestaciones del PC.

- Burbuja, Inserción y Selección
- Mergesort, Quicksort y Heapsort
- Floyd
- Hanoi

Burbuja

```
1  #include <iostream>
2  using namespace std;
3  #include <ctime>
4  #include <cstdlib>
5  #include <climits>
6  #include <cassert>
7
8  //generador de ejemplos para el problema de eliminar
    elementos repetidos. Simplemente, para rellenar el
    vector de tamaño n genera n enteros aleatorios
    entre 0 y n-1
9
10 double uniforme() //Genera un número uniformemente
    distribuido en el
11                 //intervalo [0,1) a partir de uno de
    los generadores
12                 //disponibles en C.
13 {
14     int t = rand();
15     double f = ((double)RAND_MAX+1.0);
```

Inserción

Consideramos el elemento N-ésimo de la lista y lo ordenamos respecto de los elementos desde el primero hasta el N-1-ésimo.

Es $O(n^2)$.

Selección

Consiste en encontrar el menor de todos los elementos de la lista e intercambiarlo con el de la primera posición. Luego con el segundo, y así sucesivamente hasta ordenarlo todo.

Es $O(n^2)$.

Mergesort

Se basa en la técnica de *divide y vencerás*.

- Se divide la lista a ordenar en dos sublistas de la mitad de tamaño.
- Se ordena cada sublista de forma recursiva.
- Si el tamaño de una sublista es 0 o 1 entonces ya está ordenada.
- Se unen todas las sublistas en una sola.

Es $O(n \log n)$.

Quicksort

Se basa en la técnica de *divide y vencerás*.

- Elegimos un elemento de la lista, el *pivote*.
- Se ordena la lista, dejando los elementos mayores a la derecha del pivote y los menores a la izquierda.
- Realizamos el proceso recursivamente en las dos sublistas que nos quedan (derecha e izquierda) hasta que tengan 0 o 1 elemento.

Es $O(n \log n)$.

Heapsort

Es un método de ordenación por selección.

- El *heap* es un árbol binario de altura mínima, en el que los nodos del nivel más bajo están lo más a la izquierda posible.
- Los hijos de cada nodo son siempre menores que el padre.
- No es necesario recorrer el árbol de forma desordenada para encontrar los elementos máximos.

Es $O(n \log n)$.

Floyd

Algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos ponderados.

- El algoritmo compara todos los posibles caminos a través del grafo entre cada par de vértices.

Es $O(n^3)$.

Hanoi

Las torres de Hanoi son un puzzle matemático.

- Tenemos 3 pilas: origen, auxiliar y destino.
- Origen está ordenada por tamaño creciente de discos.
- Se mueve un disco de la pila origen a la de destino si hay un único disco en la pila origen.
- Si no, se mueven todos los discos a la auxiliar, excepto el más grande.
- Por último, movemos el disco mayor al destino, y los $n - 1$ restantes encima del mayor.

Es $O(2^n)$.

Datos de la eficiencia empírica

Hemos recopilado los datos de la eficiencia empírica de la ejecución de los distintos algoritmos en varias tablas comparativas. Los datos han sido obtenidos ejecutandose en la misma máquina con Linux y sin optimización.

Hemos creado tablas para los distintos órdenes de eficiencia de los algoritmos y hemos puesto juntos aquellos que tienen el mismo.

Finalmente, para cada tabla comparativa hemos creado una gráfica.

Comparativa de los algoritmos de ordenación

Recopilando los datos de todos los algoritmos de ordenación hemos realizado una tabla comparativa en la que se muestra qué algoritmo es el más eficiente.

Comparativa según optimización y sistema operativo

Hemos elegido un representante de cada orden de eficiencia, y hemos realizado una comparativa para analizar cómo varían los tiempos de ejecución según el sistema operativo y el nivel de optimización.

Eficiencia híbrida

A continuación se recogen los gráficos que muestran tanto la eficiencia empírica como la función ajustada o *eficiencia híbrida* de cada algoritmo.

También se muestran, para cada algoritmo, las constantes ocultas en la expresión de la eficiencia teórica.

Ajustes con otras funciones

Veamos por último lo que ocurre si intentamos ajustar los datos recogidos experimentalmente a una función que no coincide con la eficiencia teórica del algoritmo.

Características de los PC donde se ha ejecutado

- Apple MacBook Pro, Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz, 8GB RAM.
Compilador: clang-800.0.38
Sistema operativo: macOS Sierra
- Dell XPS 13, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 8GB RAM.
Compilador: g++ 6.3.1
Sistema operativo: Arch Linux