Presentación práctica de eficiencia

Asignatura: Algorítmica

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

13 de abril de 2016

Índice

Introducción

Eficiencia

Divide y vencerás es una técnica algorítmica que consiste en resolver un problema diviéndolo 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.

Scripts

Script

Podemos obtener los datos fijando el número de vectores usados.

script.sh

```
g++ -std=c++11 ../src/mezcla.cpp
nelementos=10
while [ $nelementos -lt 2500 ]; do
./a.out $nelementos 200 3
let nelementos=nelementos+25
done
```

Script

Si queremos fijar el número de vectores usaremos

script.sh

```
kvectores=10
while [ $kvectores -lt 2500 ]; do
./a.out 200 $kvectores 2
let kvectores=kvectores+25
done
```

Script

Datos en 3 dimensiones, número de vectores, elementos del vector, y tiempo del algoritmo

script.sh

```
nelementos=10
nvectores=10
while [ $nelementos -lt 1000 ]; do
./a.out $nelementos 10 1
...
./a.out $nelementos 910 1
let nelementos=nelementos+100 done
```

Scripts de gnuplot

Gnuplot

Los datos están en "datos.dat". Ejecutamos algoritmo.gp

```
$ gnuplot
```

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set title "..."
set xlabel "Vectores/Elementos
del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = ...
fit f(x) "datos.dat"via a
plot "datos.dat" , f(x)
```

Funciones ajustadas

$$f(x) = a * x$$

$$g(x) = a * x * x$$

$$h(x) = a * x * (log(x)/log(2))$$

Scripts de gnuplot

Gnuplot

Los datos están en "datos.dat". Ejecutamos algoritmo.gp

```
$ gnuplot
```

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set title "..."
set xlabel "Vectores/Elementos
del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = ...
fit f(x) "datos.dat"via a
plot "datos.dat", f(x)
```

Funciones ajustadas

$$f(x) = a * x$$

$$g(x) = a * x * x$$

$$h(x) = a * x * (log(x)/log(2))$$

Scripts de gnuplot

Gnuplot

Los datos están en "datos.dat". Ejecutamos algoritmo.gp

```
$ gnuplot
```

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set title "..."
set xlabel "Vectores/Elementos del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = ...
fit f(x) "datos.dat"via a
plot "datos.dat", f(x)
```

Funciones ajustadas

$$f(x) = a * x$$

$$g(x) = a * x * x$$

$$h(x) = a * x * (log(x)/log(2))$$

Ordenador usado

Ordenador usado para la ejecución

HP Pavilion g series (Pavilion g6)
Sistema operativo: ubuntu 14.04 LTS
Memoria: 3.8 GiB (4Gb)
Procesador: Inter Core i3-2330M CPU @ 2.20GHz x 4
Gráficos: Intel Sandybridge Mobile
Tipo de SO: 64 bits
Disco: 487.9 GB

Problema

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)

Cota superior

Es posible imponer una cota superior teórica. Teniendo en cuenta que hay kn elementos, si aplicásemos un algoritmo de ordenación con eficiencia $O(n) = nlog(n)$ deducimos que podemos encontrar un algoritmo de ordenación básica con eficiencia $O(k, n) = nklog(nk)$. Tomar los k vectores como uno solo no aprovecha aún el hecho de que partes del vector están ordenadas.

Fuerza bruta

Algoritmo

En cada paso elegimos el mínimo de los primeros elementos de los k vectores, será el primer elemento del vector creciente resultante. Para el siguiente paso descartamos ese elemento y calculamos otra vez el mínimo, lo insertamos al final del vector resultante y así sucesivamente.
Buscar el mínimo es $O(k) = k$ ya que el vector de índices tiene k elementos, y lo repetimos kn veces.

Eficiencia

$$\sum_{i=1}^{kn} k = nk^2 \implies O(k, n) = nk^2$$

Divide y vencerás

Algoritmo

Usaremos mergesort, pero con los primeros montículos ya creados, por tanto tendrá una constante oculta menor que usar mergesort para kn datos arbitrarios.
En el proceso lo que haremos es ir mezclando las partes de dos en dos. El algoritmo que mezcla dos vectores en un único tiene eficiencia $O(n) = n$.

Código

```
void MergeKPartitions(int* &vector, int n_elem, int ini, int fin){
  int size = fin - ini + 1, partitions = size/n_elem;
  if(partitions == 2) // Caso base
    Merge(vector, ini, ini+n_elem, fin);
  else if(partitions > 2){
    int division = ini + (partitions/2)*n_elem; // Cálculo de la división
    MergeKPartitions(vector, n_elem, ini, division-1);  // Primer vector ordenado
    MergeKPartitions(vector, n_elem, division, fin);    // Segundo vector ordenado
    Merge(vector, ini, division, fin);  // Mezclamos los dos conjuntos
  }
}
```

Eficiencia

Donde k es el número de vectores y n el número de elementos de cada vector:s

$$T(k,n) = \begin{cases} 2n & \text{si } k = 2 \\ 2T(k/2,n) + kn & \text{si } k > 2 \end{cases}$$

Desarrollo

Sustituyendo $k = 2^m \implies T(2^m, n) = 2T(2^{m-1}, n) + 2^m n$

$$T(2^m, n) = 2\left[T(2^{m-2}, n) + 2^{m-1}n\right] + 2^m n$$

Para el caso genérico, con $j \in [0, m-1] \cap \mathbb{N}$ y desarrollando:

$$T(2^m, n) = 2^j T(2^{m-j}, n) + \sum_{i=1}^{m-1} 2^m n$$

$$T(2^m, n) = 2^{m-1} T(2, n) + \sum_{i=1}^{m-1} 2^m n$$

$$T(2^m, n) = 2^m n + (m-1)2^m n = 2^m n[1 + (m-1)] = 2^m nm$$

Eficiencia final

Solución

Deshacemos el cambio de variable, $k = 2^m \implies log_2(k) = m$:

$$T(k, n) = knlog_2 k$$