Análisis de Algoritmos 2018/2019

Práctica 2

Vuestros nombres,  Grupo.

Martín Sánchez Signorini, César Ramírez Martínez. Grupo 1201

|  |  |  |  |
| --- | --- | --- | --- |
| Código | Gráficas | Memoria | Total |
|  |  |  |  |

**1. Introducción.**

Aquí ponéis una introducción y discusión previa a la práctica.

**2. Objetivos**

Aquí indicáis el trabajo que vais a realizar en cada apartado.

2.1 Apartado 1

El objetivo de este apartado era el de implementar el famoso algoritmo de ordenación recursivo MergeSort en el fichero ordenación.c ya utilizado en la práctica anterior.

Para ello creamos dos funciones. Por un lado MergeSort, que es la función principal y la que realiza las llamadas recursivas. Y por el otro está Merge, que es la función auxiliar encargada de combinar a la vez que ordenar las subtablas que MergeSort le va pasando. Ambas funciones devuelven ERR en caso de error o el número de operaciones básicas en caso de éxito. Este hecho nos permitirá medir el rendimiento, eficacia y coste de dicho algoritmo.

Para comprobar el correcto funcionamiento del algortimo, lo probamos en el programa ejercicio4.c, reciclado de la práctica 1.

2.2 Apartado 2

En el segundo apartado hicimos uso del programa ejercicio5.c, pero con la modificación de que el algoritmo a probar ahora es MergeSort.

Gracias a este programa fuimos capaces de crear un fichero.log en el que se nos mostraba en 5 columnas diferentes:

1). El tamaño de las tablas que el algortimo iba a ordenar

2). El tiempo medio que tardaba en ordenarlas las tablas de dicho tamaño.

3). El número medio de OBS que el algoritmo hacía para tablas de dicho tamaño

4). El número máximo de OBS realizadas por el algoritmo para tablas de dicho tamaño.

5). El número mínimo deOBS realizadas por el algoritmo para tablas de dicho tamaño.

Respectivamente. Gracias a este fichero vamos a ser capaces de determinar el rendimiento del algoritmo en cuestión además de realizar las gráficas.

2.3 Apartado 3

El objetivo de este apartado es el de implementar el algoritmo de ordenación recursivo QuickSort. Para ello crearemos tres funciones en total.

Primeramente creamos QuickSort, que será la función encargada de realizar las llamadas recursivas y de llamar a otra importante función que conforma este algoritmo, Partir.

Partir será la función auxiliar encargada de ordenar los elementos de las tablas que la función QuickSort le va pasando. Para ello se apoyará en la funición Medio, que, a su vez, se encarga de elegir el pivote que Partir empleará para ordenar relativamente los elementos de la tabla. Medio, en este caso, elegirá como pivote el primer elemento de la tabla.

Luego con estas tres funciones (QuickSort, Partir y Medio) seremos capaces de implementar el algoritmo QuickSort y probaremos su correcto funcionamiento de nuevo en el progrma ejercico4.c.

2.4 Apartado 4

En este apartado realizaremos algo muy parecido a lo hecho en el apartado 2. Es decir, probando el algortimo QuickSort con el programa ejercicio5.c, seremos capaces de crear un fichero.log de la misma manera que antes. De esta forma podremos comprobar el rendimiento de este algoritmo. De nuevo este fichero será sobre los que luego nos apoyaremos para realizar las gráficas.

2.5 Apartado 5

En este apartado se nos pide que implementemos dos funciones más para elegir el pivote sobre el que la función Partir trabajará de maneras distintas. Es decir, se nos pide implementar dos variantes de la función Medio.

Por un lado tenemos la función Medio\_Avg,que simplemente elige como pivote el índice del elemento que está en la posición intermedia de la tabla.

Y por otro tenemos Medio\_Stat, una función algo más interesante. Está función escoge la posicion del valor intermedio entre el primer elemento, el último y el que ocupa la posición intermedia de la tabla.

De las tres formas de escoger el pivote que hemos visto, esta es la más eficiente. Esta es la forma en la que se tienen más probabilidades de escoger como pivote la posición de un elemento cuyo valor sea intermedio con respecto al resto de valores de los elementos de la tabla. Algo que mejora considerablemente el rendimiento del algoritmo.

**3. Herramientas y metodología**

Aquí ponéis qué entorno de desarrollo (Windows, Linux, MacOS) y herramientas habéis utilizado (Netbeans, Eclipse, gcc, Valgrind, Gnuplot, Sort, uniq, etc) y qué metodologías de desarrollo y soluciones al problema planteado habéis empleado en cada apartado. Así como las pruebas que habéis realizado a los programas desarrollados.

3.1 Apartado 1

Metodología y solución adoptada del apartado 1

3.2 Apartado 2

Metodología y solución adoptada del apartado 2

3.3 Apartado 3

Metodología y solución adoptada del apartado 3

3.4 Apartado 4

Metodología y solución adoptada del apartado 4

3.5 Apartado 5

Metodología y solución adoptada del apartado 5

**4. Código fuente**

Aquí ponéis el código fuente **exclusivamente de las rutinas que habéis desarrollado vosotros** en cada apartado.

4.1 Apartado 1

4.3 Apartado 3

4.5 Apartado 5

**5. Resultados, Gráficas**

Aquí ponis los resultados obtenidos en cada apartado, incluyendo las posibles gráficas.

5.1 Apartado 1

En este apartado fuimo capaces de comprobar el correcto funcionamiento del algoritimo MergeSort implementado gracias al programa ejercicio4.c. Como se puede apreciar en la siguiente imagen, la tabla de 20 elementos ha sido correctamente ordenada, y no se ha dado ningún tipo de pérdida de memoria.

Aquí se aprecia su correcto funcionamiento:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

5.2 Apartado 2

En este apartado utilizamos el programa ejercicio5.c para que se nos generase un fichero.log gracias al que pudimos observar con eficacia el rendimiento de MergeSort. Lo que le pedimos a MergeSort fue que nos ordenase 200 tablas de 1000 a 2000 elementos con un incremento de 100 en 100 elementos.

Imagen que contiene texto, captura de pantalla, mapa

Descripción generada automáticamente

En esta gráfica podemos apreciar el número máximo (en azul), número medio (en rosa) y el número mínimo (en verde) de OBS que realiza el algortimo para tablas de tamaño creciente hasta 2000 elementos por tabla. También hemos añadido la gráfica de la función 1,25\*x\*log(x), que se ajusta bastante bien a las anteriores.

Este último hecho nos hace darnos cuenta del correcto funcionamiento del algoritmo. Ya que como sabemos el caso mejor, medio y peor de este es de O(Nlog(N)).

Imagen que contiene mapa, texto

Descripción generada automáticamente

En esta gráfica se aprecia el tiempo medio que tarda MergeSort en ordenar tablas de 1000 a 2000 elementos. Vemos que es bastante coherente al ir creciendo a medida que también lo hace el tamaño de las tablas.

5.3 Apartado 3

En este apartado fuimo capaces de comprobar el correcto funcionamiento del algoritimo QuickSort implementado gracias de nuevo al programa ejercicio4.c. Como se puede apreciar en la siguiente imagen, la tabla de 20 elementos ha sido correctamente ordenada, y no se ha dado ningún tipo de pérdida de memoria.

Aquí se aprecia su correcto funcionamiento:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

5.4 Apartado 4

En este apartado utilizamos el programa ejercicio5.c para que se nos generase un fichero.log gracias al que pudimos observar con eficacia el rendimiento de QuickSort. Lo que le pedimos a QuickSort fue que nos ordenase 200 tablas de 1000 a 2000 elementos con un incremento de 100 en 100 elementos.

Imagen que contiene texto, mapa

Descripción generada automáticamente

En esta gráfica podemos apreciar el número máximo (en azul), número medio (en rosa) y el número mínimo (en verde) de OBS que realiza el algortimo para tablas de tamaño creciente hasta 2000 elementos por tabla. También hemos añadido la gráfica de la función 1,75\*x\*log(x), que se ajusta bastante bien a las gráficas azul (máximo de OBS) y rosa (media OBS).

Este último hecho nos hace darnos cuenta del correcto funcionamiento del algoritmo. Ya que como sabemos el caso mejor y medio de este es de O(Nlog(N)).

Imagen que contiene mapa, texto

Descripción generada automáticamente

En esta gráfica se aprecia el tiempo medio que tarda QuickSort en ordenar tablas de 1000 a 2000 elementos. Vemos que es bastante coherente al ir creciendo a medida que también lo hace el tamaño de las tablas.

5.5 Apartado 5

En este apartado compararemos mediante una gráfica las diferencias existentes entre el funcionamineto y rendimiento de QuickSort con las diferentes variantes de la función Medio que implementamos antes. En este caso le pedimos a Quicksort que nos ordenase 1000 tablas de 1000 a 10000 elementos con un incremento de 500 en 500 elementos.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

La gráfica rosa se corresponde con el número medio de OBS que realiza QuickSort para ordenar tablas (de tamaño 1000 a 10000) con la función Medio inicial, la que simplemente elegía como pivote el primer elemento de la tabla.

La gráfica verde se corresponde con el número medio de OBS que realiza QuickSort para ordenar tablas (de tamaño 1000 a 10000) con la función Medio\_Avg. Esta elegía como pivote el elemento que se encontraba justo en la posición intermedia de la tabla.

Y por último tenemos la gráfica azul, que se corresponde con el número medio de OBS que realiza QuickSort para ordenar tablas (de tamaño 1000 a 10000) con la función Medio\_Stat. Esta comparaba los valores de los elemento incial, intermedio y último de la tabla, y elegía como pivote el elemento cuyo valor fuese el intermedio de los tres.

Como es de esperar es esta última la función que mejor rendimiento da al algoritmo. Esto se debe a que la manera más rápida de ordenar una tabla es eligiendo como pivote el elemento cuyo valor es el intermedio de todos los valores de los elementos de la tabla. Y esta función es la que más se acerca a realizar exitosamente esta tarea.

Las otras dos variantes de Medio, son exactamente igual de eficientes.

**5. Respuesta a las preguntas teóricas.**

Aquí respondéis a las preguntas teóricas que se os han planteado en la práctica.

5.1 Pregunta 1

5.2 Pregunta 2

5.3 Pregunta 3

5.4 Pregunta 4

5.5 Pregunta 5

**6. Conclusiones finales.**

Este práctica nos ha hecho darnos cuenta de una manera práctica de todos lo contenidos explicados en el tema 2 en clases teóricas.

Hemos sido capaces de observar y entender con mucho mayor detalle cómo funcionan algunos algoritmos de ordenación recursivos y mucho más eficientes que los anteriores vistos como son MergeSort e QuickSort y, sobre todo, cómo calcular su rendimiento y eficacia.

Gracias a las gráficas hemos podido comparar los números de OBs que hacían nuestros algoritmos para permutaciones de diferentes tamaños y entender cómo estos funcionaban en realidad. Otra cosa que llama la atención es cómo, de manera práctica, se puede observar que, en cuanto al rendimiento de ambos algoritmos (QuickSort e MergeSort) para ordenar tablas, no existe demasiada diferencia en cuanto OBs se refiere. Pero ni nos fijamos en el tiempo que tarda cada algoritmo en ordenar y no en las OBs, te das cuenta de que QuickSort es bastante más rápido. Además de que este requiere usar mucha menos memoria de la que lo hace MergeSort. Al utilizar herramientas como Valgrind te das cuenta rápidamente de este hecho.

Hemos concluido en esta práctica que claramente el uso de algortimos recursivos frente a los no recursivos mejora muy considerablemente el rendimiento y el tiempo que estos tardan en ordenar las permutaciones.

Imagen que contiene texto

Descripción generada automáticamente

Imagen que contiene texto, captura de pantalla

Descripción generada automáticamente

Para comprobar si los resultados experimentales de los casos peor, mejor y medio se asemejaban a los teóricos, nos generamos 1000000 de tablas de 1 a 10 elementos, para intentar obtener todas las posibles permutaciones. A contunuación las ordenamos con ambos algortimos. Y, a finalmente veríamos si lo que obtuvimos concuerda con lo explicado en clase. Aquí se aprecia la diferencia en cuanto al uso de memoria se refiere.

Estas fueron las gráficas obtenidas:

-Para MergeSort:

Imagen que contiene mapa, texto

Descripción generada automáticamente

-Para QuickSort:

Imagen que contiene mapa, texto

Descripción generada automáticamente

Y estos fueron los ficheros.log generados:

-Para MergeSort:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Un ejemplo claro de que estos resultados se asemejan con los teóricos, es el caso de, por ejemplo, las tablas de N = 9 elementos, en las que se aprecia un caso mejor de 13 OBs (0,46\*N\*), uno peor de 21 OBs (0,74\*N\*)y uno medio de 19 (0,67\*N\*). Luego todos los casos son del orden de O(N\*)), tal y cómo esperábamos.

-Para QuickSort:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Un ejemplo en este caso de que estos resultados se asemejan con los teóricos, es el caso de, por ejemplo, las tablas de N = 9 elementos, en las que se aprecia un caso mejor de 24 OBs (0,84\*N\*), uno peor de 35 OBs () y uno medio de 29 (N\*). Luego los casos medio y mejor son del orden de O(N\*)), y el peor del O(tal y cómo esperábamos.

Gracias a los ficheros.log, se puede observar facilmente el hecho de que MergeSort realiza menos OBs para ordenar las misma tablas que QuickSort, pero, en cambio, este último es bastante más rápido (en cuanto a tiempo se refiere), que es lo que realmente interesa al final.