Análisis de Algoritmos 2017/2018

Práctica 2

Victoria Pelayo e Ignacio Rabuñal,  Grupo 1273.

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

**1. Introducción.**

El objetivo que se nos ha propuesto en esta práctica es el analizar dos conocidos algoritmos que siguen la metodología de divide y vencerás: MergeSort y QuickSort.

A lo largo de la práctica deberemos implementar ambos algoritmos y analizar su rendimiento para poder comparar los resultados con los supuestos teóricos.

**2. Objetivos**

2.1 Apartado 1

En este apartado debemos crear las funciones “merge” y “mergesort” que serán necesarias para seguir el algoritmo de ordenación MergeSort. Una vez creadas debemos probar con el programa ejercicio4.c que el algoritmo ordena correctamente.

2.2 Apartado 2

El objetivo de este apartado es modificar el archivo ejercicio5.c para obtener las tablas de tiempo y operaciones básicas de MergeSort. Una vez obtenidas las tablas, serán representadas para comparar los resultados con los supuestos teóricos.

2.3 Apartado 3

En este apartado debemos implementar el algoritmo QuickSort. Para ello, crearemos primero las funciónes “medio” y “partir” que se utilizaran en la implementación del algoritmo.

2.4 Apartado 4

El objetivo de este apartado es modificar el archivo ejercicio5.c para obtener las tablas de tiempo y operaciones básicas de QuickSort. Una vez obtenidas las tablas, serán representadas para comparar los resultados con los supuestos teóricos.

2.5 Apartado 5

En este último apartado debemos implementar dos funciones “medio\_avg” y “medio\_stat” que eligen el pivote para QuickSort de manera diferente. Consecuentemente, deberemos modificar la función “partir” para que sea posible usar cualquier función de elección de pivote. Para acabar, compararemos los tiempos de ejecucion y promedio de Obs de QuickSort con las diferentes rutinas pivote.

**3. Herramientas y metodología**

En todos los apartados hemos utilizado el entorno Linux, el editor Atom para crear las funciones necesarias o modificar archivos anteriores. A la hora de comprobar nuestos ficheros, hemos utilizado gcc para compilar los archivos, Valgrind para asegurar una correcta gestión de memoria y el programa Gnuplot para representar los resultados obtenidos.

3.1 Apartado 1

Para empezar implementamos la función “merge” siguiendo el pseudocódigo propuesto en las transparencias de teoría y a continuación creamos “mergesort” siguiendo la misma metodología. Por último comprobamos la corrección de nuestro código con el programa ejercicio4.c.

3.2 Apartado 2

En el apartado anterior creamos el algoritmo MergeSort con un contador de OBs por lo que después de la ligera modificación de ejercicio5.c para adaptarlo a MergeSort y medimos los tiempos y OBs sin problemas.

3.3 Apartado 3

En primer lugar, implementamos la función “medio” trivialmente. A continuación nos servimos de los pseudocódigos de las transparencias de teoría de la asignatura para construír las funciones “partir” y “QuickSort”. Por último, modificamos el archivo ejercicio4.c para que sea aplicable al algoritmo creado previamente y comprobamos su corrección.

3.4 Apartado 4

Realizamos una ligera modificación en ejercicio5.c para adaptarlo a “QuickSort” y medimos los timepos y OBs sin problemas.

3.5 Apartado 5

La función “medio\_avg” fue adaptada trivialmente a partir de la función medio y para crear la función “medio\_stat” nos ayudamos de un árbol de decisión para representar la elección del pivote. Por último, adaptamos la función “partir” y “QuickSort” añadiéndole un puntero a función para escoger el método de elección de pivote.

También creamos tres funciones en ordenación.c que solo reciben tres parámetros y estas serán las que pasaremos en ejercicio5.c para guardar en un fichero los tiempos y el número de ejecuciones de OBs de “QuickSort” con sus distintintas funciones para elegir el pivote.

**4. Código fuente**

4.1 Apartado 1

int mergesort(int\* tabla, int ip, int iu){

int M = (iu + ip)/2;

int count = 0;

if (ip == iu) return count;

count += mergesort(tabla, ip, M);

count += mergesort(tabla, M + 1, iu);

count += merge (tabla, ip, iu, M);

return count;

}

int merge(int\* tabla, int ip, int iu, int imedio){

int \*aux = NULL;

int i = ip;

int k = 0;

int j = imedio +1;

int count = 0;

aux = (int\*)malloc(sizeof(aux[0]) \* (iu - ip + 1));

if(aux == NULL) return ERR;

while (i <= imedio && j<= iu){

count++;

if (tabla[i] < tabla[j]){

aux[k] = tabla[i];

i++;

}

else{

aux[k] = tabla[j];

j++;

}

k++;

}

if(i > imedio){

while(j <= iu){

aux[k] = tabla[j];

k++;

j++;

}

}

else if(j > iu){

while( i <= imedio){

aux[k] = tabla[i];

k++;

i++;

}

}

tabla+=ip;

for(i = 0; i < k; i++){

tabla[i] = aux[i];

}

tabla-=ip;

free(aux);

return OK;

}

4.3 Apartado 3

int medio(int \*tabla, int ip, int iu,int \*pos){

\*pos = ip;

return 0;

}

int partir(int\* tabla, int ip, int iu, int \*pos, pfunc\_ordena metodo){

int k;

int i;

int ob = 0;

\*pos = ip;

k = tabla[\*pos];

swap(&tabla[ip], &tabla[\*pos]);

\*pos = ip;

for(i = ip + 1; i <= iu; i++ ){

ob++;

if(tabla[i] < k){

(\*pos)++;

swap(&tabla[i], &tabla[\*pos]);

}

}

swap(&tabla[ip], &tabla[\*pos]);

return ob;

}

int quicksort(int\* tabla, int ip, int iu){

int pos = 0;

int ob = 0;

if(ip > iu) return ERR;

else if(ip == iu) return ob;

else{

ob += partir(tabla, ip, iu, &pos, Mediana);

if( ip < pos - 1)

quicksort(tabla, ip, pos - 1);

if( pos + 1 < iu)

quicksort(tabla, pos + 1, iu);

}

return ob;

}

4.5 Apartado 5

int medio\_avg(int \*tabla, int ip, int iu, int \*pos){

\*pos = (ip + iu) / 2;

return 0;

}

int Mediana (int\* tabla, int ip, int iu, int\* pos){

int im, ob;

ob++;

if (tabla[ip] <= tabla[im]){

ob++;

if(tabla[iu] <= tabla[im]){

ob++;

if(tabla[ip] <= tabla[iu]){

\*pos = iu;

}

else \*pos = ip;

}

else \*pos = im;

}

else {

ob++;

if(tabla[im] <= tabla[iu]){

ob++;

if(tabla[iu] <= tabla[ip]){

\*pos = iu;

}

else \*pos = ip;

}

else \*pos = im;

}

return ob;

}

**5. Resultados, Gráficas**

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

5.1 Apartado 1

Resultados del apartado 1.

5.2 Apartado 2

Resultados del apartado 2.

Gráfica comparando los tiempos mejor peor y medio en OBs para MergeSort, comentarios a la gráfica.

Gráfica con el tiempo medio de reloj para MergeSort, comentarios a la gráfica.

5.3 Apartado 3

Resultados del apartado 3.

5.4 Apartado 4

Resultados del apartado 4.

Gráfica comparando los tiempos mejor peor y medio en OBs para QuickSort, comentarios a la gráfica.

Gráfica con el tiempo medio de reloj para QuickSort, comentarios a la gráfica.

5.5 Apartado 5

Resultados del apartado 5.

Gráfica con el tiempo medio de reloj comparando los tres pivotes empleados, comentarios a la gráfica.

**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

**6. Conclusiones finales.**

Discusión final sobre la práctica y los resultados obtenidos.