Práctica 7

Tulio Muñoz Magaña

Marco teórico

El algoritmo Merge Sort es un algoritmo de ordenamiento de complejidad `O(n log_2 n)` en el peor caso, la cual es una mejoría considerable relativa a la complejidad de algoritmos como selection sort o insertion sort, cuyas complejidades en el peor caso son `O(n^2)`. El algoritmo usa el paradigma de programación Divide and Conquer, ya que divide el problema en subproblemas y resuelve éstos por separado, para luego juntar las soluciones y generar la solución del problema original. La idea principal tras el algoritmo es dividir el arreglo en mitades, y una vez que se ordenen, mezclar ambas mitades elemento por elemento hasta generar el arreglo ordenado. Este algoritmo es estable pero no se ordena en sitio.

El algoritmo Quick Sort es un algoritmo de ordenamiento que también utiliza el paradigma de programación Divide and Conquer. La forma en que Quick Sort ordena el arreglo es escoger un elemento, al cual le llamamos el pivote, y colocar los elementos menores al pivote a la izquierda de éste y los mayores a la derecha. A este proceso se le llama proceso de partición. Al realizar este procedimiento recursivamente se llega al arreglo ordenado. El algoritmo es estable y tiene complejidad `O(n^2)` en el peor caso, sin embargo, posee complejidad `O(n log_2 n)` en el caso promedio. Éste último detalle es la razón por la que en ocasiones se prefiere el algoritmo Quick Sort al Merge Sort, ya que la complejidad en el caso promedio es igual a la complejidad del peor caso de Merge Sort, y sin embargo Quick Sort tiene las ventajas de que se ordena en sitio y la constante por la cual tiene complejidad `O(n^2)` es menor a la constante por la cual Merge Sort tiene complejidad `O(n log_2 n)`.

Descripción de los programas

En el archivo `ejercicio1.cpp` se ha creado una implementación de Merge Sort. La función recibe el arreglo a ordenar, los índices del principio y el final en donde se ordenará, y una función para comparar dos elementos del mismo tipo que los elementos del arreglo. Se espera que esta función de comparación regrese 1 si el primer elemento que recibe es "menor" que el segundo, y 0 en otro caso. El algoritmo no regresa nada, deja el arreglo ya ordenado en su lugar.

En el archivo `ejercicio2.cpp` se ha creado una implementación de Quick Sort. La función recibe el arreglo a ordenar, los índices del principio y el final en donde se ordenará, y una función para comparar dos elementos del mismo tipo que los elementos del arreglo. Se asume lo mismo para esta función de comparación que para la solicitada en el ejercicio 1. El algoritmo no regresa nada, deja el arreglo ya ordenado en su lugar.

En el archivo `ejercicio3.cpp` se realizan experimentos para probar los algoritmos de ordenamiento implementados en los ejercicios 1 y 2. Primero se crean 2 arreglos, uno de enteros y otro de dobles. Se imprimen en pantalla los arreglos y luego se ordena cada uno de manera ascendente con Merge Sort, de manera ascendente con Quick Sort y de manera descendente también con ambos algoritmos. Después de realizar cada ordenamiento se imprime el resultado en pantalla.

Además se crea una estructura `pareja` que representa un par ordenado con coordenadas `x` y `y`. Se genera un arreglo de parejas de enteros y se imprime en pantalla. Se crean dos funciones de comparación para parejas, una que compara la primera coordenada de ambas y otra que compara la segunda. Se ordena el arreglo de

parejas tanto por la primera como por la segunda coordenada y con ambos algoritmos, se imprimen los resultados después de cada ordenamiento.

El programa no recibe ninguna entrada.

Los comandos de compilación y ejecución respectivamente son:

```
g++ -std=c++11 ejercicio3.cpp
./a.out
```

La salida del programa es la siguiente:

```
El arreglo de enteros original:
1 5 -4 7 -23 25 76 2 4 -9 0

El arreglo de enteros ordenado ascendentemente por Merge Sort:
-23 -9 -4 0 1 2 4 5 7 25 76
El arreglo de enteros ordenado ascendentemente por Quick Sort:
-23 -9 -4 0 1 2 4 5 7 25 76

El arreglo de enteros ordenado descendentemente por Merge Sort:
76 25 7 5 4 2 1 0 -4 -9 -23
El arreglo de enteros ordenado descendentemente por Quick Sort:
76 25 7 5 4 2 1 0 -4 -9 -23

El arreglo de doubles original:
2.54 7.68 -2.3 12 -67.8 34.9 3 -5.3

El arreglo de doubles ordenado ascendentemente por Merge Sort:
-67.8 -5.3 -2.3 2.54 3 7.68 12 34.9
El arreglo de doubles ordenado ascendentemente por Quick Sort:
-67.8 -5.3 -2.3 2.54 3 7.68 12 34.9

El arreglo de doubles ordenado descendentemente por Merge Sort:
34.9 12 7.68 3 2.54 -2.3 -5.3 -67.8
El arreglo de doubles ordenado descendentemente por Quick Sort:
34.9 12 7.68 3 2.54 -2.3 -5.3 -67.8

El arreglo de parejas original:
(4,2) (23,8) (-2,9) (3,0) (-12,-7)

El arreglo de parejas ordenado por la primera coordenada por Quick Sort:
(-12,-7) (-2,9) (3,0) (4,2) (23,8)
El arreglo de parejas ordenado por la primera coordenada por Merge Sort:
(-12,-7) (-2,9) (3,0) (4,2) (23,8)

El arreglo de parejas ordenado por la segunda coordenada por Quick Sort:
(-12,-7) (3,0) (4,2) (23,8) (-2,9)
```

```
El arreglo de parejas ordenado por la segunda coordenada por Merge Sort:
(-12,-7) (3,0) (4,2) (23,8) (-2,9)
```

La captura de pantalla de este algoritmo corriendo se llama `captura.png`.

Resultados obtenidos

Se ha logrado imlementar correctamente los algoritmos de ordenamiento Merge Sort y Quick Sort, así como permitir flexibilidad en los tipos de datos que se ordenan y el criterio de ordenamiento, al hacerse este ordenamiento con la función de comparación que da el usuario.

Conclusión

Los algoritmos Merge Sort y Quick Sort poseen sus ventajas cada uno. Ambos algoritmos son estables y tienen complejidad baja comparada con otros algoritmos de ordenamiento, por lo que son muy buenas opciones. La decisión entre usar uno u otro estribará en el tipo de problema a resolver, en cuya situación consideraremos si se requiere que el ordenamiento se haga en sito, o si la cantidad de elementos es excesivamente alta. De cualquier forma, es útil poseer ambos algoritmos en el repertorio.

Referencias

- https://www.cplusplus.com/doc/oldtutorial/templates/
- https://www.codespeedy.com/copy-elements-of-one-array-to-another-in-cpp/
- Material de la clase