Proyecto 1. Ordenamiento externo

Calderón Jiménez, David
Hernández Olvera, Humberto Ignacio
Ordiales Caballero, Iñaky

15/Noviembre/2020

Objetivo

Que el alumno implemente los algoritmos de ordenamiento externo, que conozca elementos para el manejo de archivos, aplique los conceptos generales de programación y desarrolle sus habilidades de trabajo en equipo.

Introducción

¿Por qué ordenar?[1]

Muchos científicos de la computación consideran que ordenar es el problema más fundamental en el estudio de los algoritmos. Algunas razones son:

- En ocasiones la necesidad de ordenar información es inherente en una aplicación. Por ejemplo, en un banco, se necesitan ordenar por números de cheque cuando se requiere preparar estados de cuenta de un cliente.
- Los algoritmos suelen usar el ordenamiento como una subrutina clave.
- Muchos problemas de ingeniería salen a flote cuando se implementan algoritmos de ordenamiento. El programa de ordenamiento más rápido para una situación en particular depende de muchos factores: el conocimiento previo acerca de las claves y datos de satélite, la jerarquía de memoria de la computadora host y el ambiente de software, por mencionar algunos.

El principal objetivo del ordenamiento es facilitar la operación de búsqueda en una colección de datos. Se organiza siguiendo una lógica o un criterio para establecer una secuencia de acuerdo a una regla predefinida. La clave para analizar la eficiencia de un algoritmo es identificar las operaciones fundamentales: Comparación, Inserción, Intercambio, Intercalación. (Clase EDA2, 23/Septiembre/2020)

Una clasificación de los algoritmos de ordenamiento es aquella que los jerarquiza de acuerdo a la forma en la que utiliza la memoria en la computadora.

- Ordenamiento interno. El algoritmo trabaja sobre la memoria principal RAM.
- Ordenamiento Externo. El algoritmo trabaja sobre memoria secundaria (archivos).

En este proyecto se trabajará con algoritmos de ordenamiento externo. Para estos casos, ordenar un archivo implica leer su contenido, procesarlo y escribirlo nuevamente. Esto se hace a través de las siguientes operaciones:

- Dividir el archivo a ordenar en bloques de tamaño manipulable en memoria principal.
- Ordenar los bloques y colocarlos en archivos auxiliares.
- Hacer una nueva escritura mediante la mezcla de los bloques ya ordenados.

La eficiencia de estos algoritmos está determinada por el tiempo de acceso al dispositivo externo. Los algoritmos que se utilizan en este proyecto son:

- Polifase
- Mezcla equilibrada
- Método por distribución o Radix externo

Polifase

Se compone de dos fases principalmente.

Fase 1. Lectura y división del archivo original

1. Leer una cantidad 'n' de claves/valores.
2. Utilizar un algoritmo de ordenamiento interno para ordenarlos.
3. Generar un 'bloque' con esas claves y colocarlos en un archivo auxiliar 1.
4. Repetir los pases 1 a 3 y colocar el nuevo bloque en un segundo archivo auxiliar 2.
5. Los siguientes bloques generados se intercalan entre los archivos.

Fase 2. Ordenamiento y salida

1. Intercalar el primer bloque del archivo auxiliar 1 con el primer bloque del archivo auxiliar 2 y dejar el resultado en el archivo original.
2. Intercalar el siguiente bloque de cada archivo y dejar el resultado en un tercer archivo auxiliar 3.
3. Repetir 1 y 2 hasta que no se tengan más claves para procesar.

Finalmente, se repiten las fases 1 y 2 hasta que se tengan todas las claves ordenadas.

Mezcla Equilibrada

Consiste en la realización sucesiva de particiones tomando secuencias ordenadas de máxima longitud y la mezcla de estas particiones para producir secuencias ordenadas.

1. Se busca en el archivo original por claves ya ordenadas desde el principio, cuando se encuentra un elemento que no esté ordenado con respecto a los anteriores, se toma el 'bloque' ordenado y se escribe en un archivo auxiliar 1.
2. Se busca un nuevo 'bloque' ordenado y se escribe en un archivo auxiliar 2.
3. Se repiten los pasos 1 y 2 hasta que se termine de leer el archivo.
4. Se intercalan el primer bloque del archivo 1 con el primer bloque del archivo 2, se ordenan y se escriben en el archivo original. Esto se hace con todos los bloques de los archivos 1 y 2.

Finalmente, se repiten los pasos 1 a 4 hasta que se tengan las claves ordenadas y escritas en el archivo original.

Método por distribución o Radix externo

El ordenamiento por el método de distribución, más en específico por el método de Radix Sort externo, toma como base el algoritmo de ordenamiento interno del mismo nombre. En él la idea básica para llevar a cabo el ordenamiento es ir revisando cada dígito significativo de la clave a ordenar y distribuirla en una serie de colas de todas las opciones que puede llegar a tomar el dígito. En el caso numérico se contaría con 10 colas del 0 al 9 y en el caso de letras se debería contar con 27 colas una por cada letra del abecedario en español. En cada iteraciónn se clasifican en sus colas correspondientes todas las claves y posteriormente se vacían empezando con la de menor valor en la estructura o archivo original.

La diferencia fundamental del radix externo con el interno es que en lugar de utilizar estructuras colas para cada posible valor del dígito, radix externo utiliza archivos auxiliares que harán la función de las colas, pero sin estar en memoria de ejecución. Radix Sort es un algoritmo con una muy buena complejidad de tiempo, siendo casi constante. Sin embargo la necesidad de usar colas auxiliares o inclusive archivos auxiliares hacen que tenga una complejidad espacial bastante mala. Por lo que se debe de analizar cuando su uso es realmente eficiente y cuando se estaría desaprovechando espacio por ahorrarnos un poco de tiempo.

Desarrollo

Se procedió a organizar y distribuir el trabajo, al ser 3 algoritmos de ordenamiento y 3 integrantes del equipo, se decidió entonces que cada uno desarrollaría un algoritmo, compartiendo con los demás sus avances, dudas y problemas para que todos estuvieran al tanto y ayudar en el desarrollo del programa.

Polifase

Como vimos este algoritmo externo consta de 2 partes muy importantes, la cual es un ordenamiento interno y la intercalación con los archivos externos.

La primer parte esta conformada por la primer lectura de archivos más un ordenamiento interno el cual decidimos que seria BubleSort este se decidió a partir de que es uno de los mas sencillos de implementar y ya que la implementación es con objetos pues este recibe unas modificaciones muy concretas para que este funcionara en objetos.

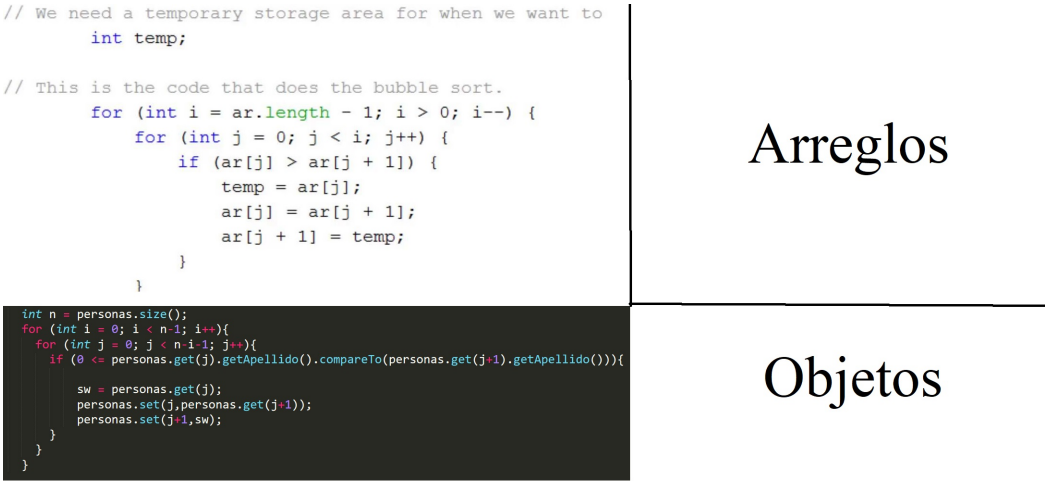

Figura 1: BubbleSort en arreglos y objetos.

Esta parte nos dará como resultado un conjunto de bloques de 4 elementos pues este tamaño fue el que elegimos para el algoritmo.

Después de esta parte solo quedaba escribir estos bloques en el archivo original.

La segunda parte solo constaba en hacer la mezcla de los datos leídos de los archivos, para que de esta forma se ordenaran los datos poco a poco.

Es decir si tengo bloques de 4 elementos pues para la siguiente iteración generaremos bloques de 8 y después de 16 y así hasta generar un bloque que contenga a todos los elementos ordenados.

Por ejemplo con 16 elementos quedaría de la siguiente manera:

Iteración 1:
G0=(Carlos,Paco,Edgar,David,Ruben,Oliver,Jose,Luis,Linda,Anei,Abigail,Toño,Raul,Kevin,Polo,Brenda)
G1=(Carlos,David,Edgar,Paco)(Abigail,Anei,Linda,Toño)
G2=(Jose,Luis,Oliver,Ruben)(Brenda,Kevin,Polo,Raul)

Iteración 2:
G0=(Carlos,David,Edgar,Jose,Luis,Oliver,Paco,Ruben)(Abigail,Anei,Brenda,Kevin,Linda,Polo,Raul,Toño)
G1=(Carlos,David,Edgar,Jose,Luis,Oliver,Paco,Ruben)
G2=(Abigail,Anei,Brenda,Kevin,Linda,Polo,Raul,Toño)

Final:
G0=(Abigail,Anei,Brenda,Carlos,David,Edgar,Jose,Kevin,Linda,Luis,Oliver,Paco,Polo,Raul,Ruben,Toño)

Como vemos la intercalación puede ser de lo mas útil al momento de ordenar, pues con esta se evitan muchas iteraciones las cuales harán que el trabajo sea mas tardado.

Al final este algoritmo nos mostrara una lista la cual estará ordenada ya sea por nombre o por apellido.

Mezcla Equilibrada

Como se mencionó antes, en mezcla equilibrada buscamos bloques de tamaño variable en donde se tienen elementos ya ordenados, y los separamos en archivos diferentes, para después regresarlos al archivo original y ahí hacer el intercalamiento de los bloques.

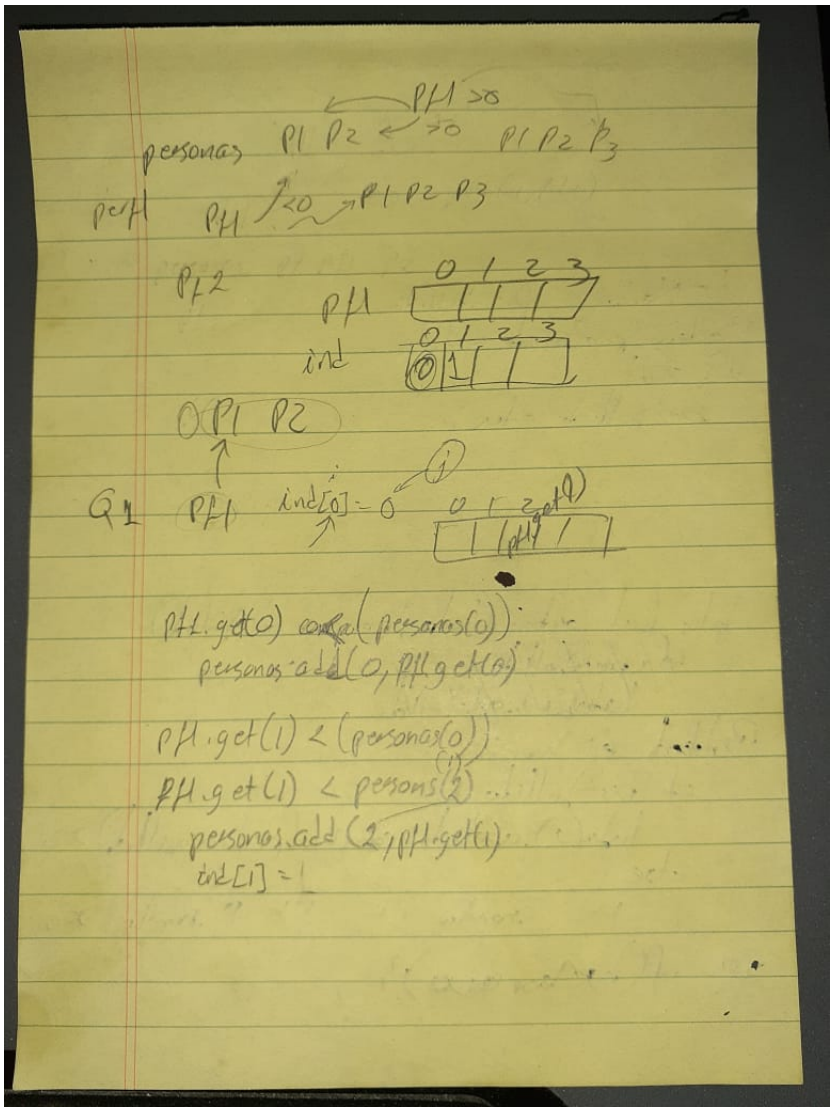

Figura 2: Primer aproximación para la resolución.

En la imagen se observa un primer boceto de como hacer el intercalamiento de las personas, haciendo la comparación e ingresarlo en el índice que le corresponde.

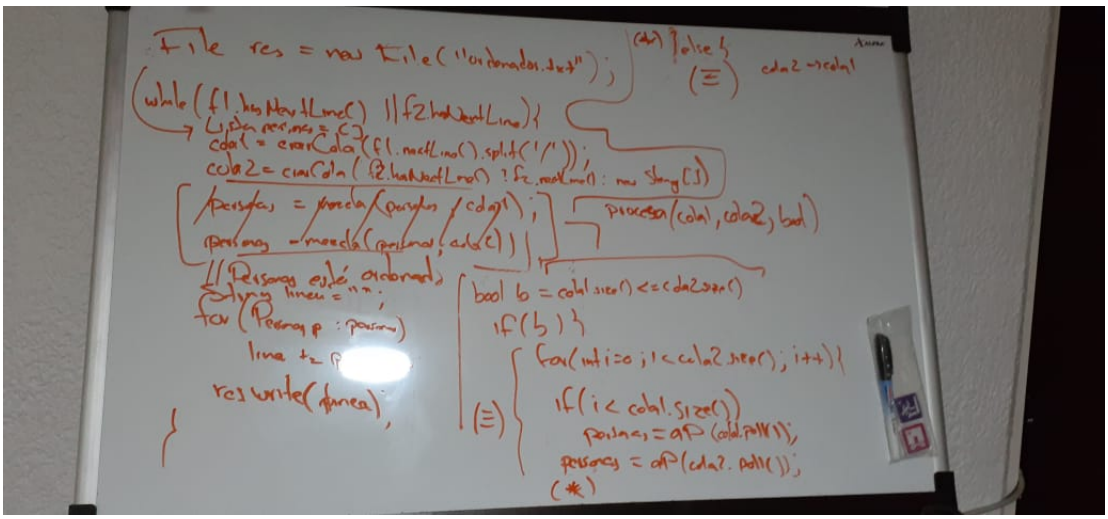

Figura 3: Pseudocódigo de lectura e intercalación de los bloques.

En la imagen vemos otro boceto pero ahora en pseudocódigo del método central para mezcla equilibrada, la lectura de cada línea de F1 y F2 para intercalarlos y escribirlos en F0.

Primero se escriben sobre F1 y F2 los bloques ordenados mediante el método *ordenaI1*, en todo momento, cuando se requiere hacer el ordenamiento de los bloques, cada persona que se lee de un archivo, se crea un objeto de tipo Persona para así poder manejar un solo objeto en lugar de 3 (que es la cantidad de atributos que tiene la persona), y con esto utilizar el método *toString* para tener un estándar de escritura en los archivos. Este método solo se utiliza al principio del ordenamiento.

El método *escribeArchivoFinal* escribe en el archivo ya sea F1 o F2, la última lista de personas que se tiene guardada en memoria, ya que al haber llegado al final de F0, no entra a la comparación normal puesto que ese último bloque está ordenado y no se pudo escribir dentro del ciclo de lectura-comparación-escritura, por lo que se debe escribir por a parte.

Con el método *ordenaI2* se lee una línea de F1 y de F2, creamos una cola de personas para los registros de F1 y otra para los registros de F2 con el método *creaCola*. En una lista de personas es en donde se tendrán ordenadas las personas de ambas colas, mediante el método *procesaColas*, si la cola menor está vacía sólo se agregan las personas de la cola mayor a la lista de personas y se regresa dicha lista. Esta situación sólo se da cuando el archivo F1 tiene una línea más que el archivo F2.

Dentro de *procesaColas* ciclamos a lo largo de la cola mayor, y mientras el iterador sea menor que el tamaño de la cola menor, se manda llamar al método *agregaPersona* con la persona de la cola menor, y posteriormente con la persona de la cola mayor. El método *agregaPersona* primero saca un elemento de la cola de menor tamaño y se agrega a la lista, de la segunda cola sacamos al primer elemento y lo comparamos, si es menor se agrega al principio, si no al final. Después, sacamos el primer elemento de la primera cola y se compara con el primero de la lista, si es menor se agrega ahí, si no, en una lista auxiliar ponemos al primero de la lista de personas, y volvemos a comparar, ahora con el nuevo primero de la lista, y así sucesivamente hasta que se encuentre su ubicación final. Después la lista de personas se escribe en F0 y se repite el procedimiento hasta que se leen todas las líneas de F1 y F2. Este método será iterativo y se repite mientras no tengamos los registros ordenados.

Con el método *ordenaI3* volvemos a escribir estos nuevos bloques de mayor tamaño en F1 y F2 alternadamente, en este método no se crean listas o colas de personas, solamente se copia cada línea del archivo F0 a su archivo destino, ya que cada línea representa un bloque que ya está ordenado.

Una consideración que se tuvo que hacer, es que se rompiera el ciclo cuando F1 y F2 tuvieran exactamente una línea, así mandar llamar al método que escribe en F0 desde F1 y F2 intercalando las líneas y ordenándolas.

El método *escribeArchivoFinal* escribe en el archivo ya sea F1 o F2, la última lista de personas que se tiene guardada en memoria, ya que al haber llegado al final de F0, no entra a la comparación normal puesto que ese último bloque está ordenado y no se pudo escribir dentro del ciclo de lectura-comparación-escritura, por lo que se debe escribir por a parte.

Para terminar, se debe regresar el archivo a su formato original, para eso está el método *regresaAFormato*.

Finalmente, en pantalla se le avisa al usuario en donde se encuentran los archivos con el acumulado de las iteraciones para que pueda verlos y revisarlos, haciendo la distinción sobre si se ordenó por apellido o por nombre, además del número de iteración, por ejemplo, si se hicieron 3 ordenamientos por nombre, cada resultado estará en una carpeta distinta dentro de la carpeta *mezclaequilibrada* cada una propiamente identificada con su número.

Todas las clases relacionadas a este algoritmo se encuentran en la carpeta de *mezclaequilibrada/codigos*.

Radix Externo

El desarrollo del algoritmo de Radix externo no presentó tantas complicaciones como los dos anteriores. La naturaleza del algoritmo y el hecho de que su implementación sólo tuviera que ser estrictamente para el ordenamiento de los datos a través de la clave de número de cuenta, hicieron que no tuviera el mismo nivel de dificultad que los algoritmo Polifase y Mezcla los cuales funcionan para los dos criterios de ordenamiento (nombre, apellido). Sin embargo no por ser más sencillo significa que no se necesitó hacer una implementación limpia y cuidadosa. A continuación se hablará del desarrollo de este algoritmo.

Lo primero que se hizo fue pensar en cómo se debía trabajar con los archivos ya que estos son la base esencial del algoritmo de ordenamiento externo. Para esto se decidió que los archivos equivalentes a las colas se crearían y destruirían durante la ejecución del programa. Para esto se crearon métodos crearArchivosColas el cual creaba tanto la carpeta 'Temporal' que es donde se ubicarían estos archivos durante el algoritmo como cada uno de los archivos. Para que se borraran al terminar el algoritmo se empleo el método deleteOnExit() para archivos. Además se tiene un método limpiarColas() el cual se encarga de borrar el contenido de los archivos entre cada iteración. Por el otro lado, los archivos que sí queríamos que se guardarán son dos. El archivo que se deseaba ordenar sobre escrito y ya ordenado. Y el archivo que llevara el registro de como iba quedando el orden después de cada iteración. El primero se decidió que se haría sobre el mismo archivo que nos diera el usuario, por lo que como se dice en el manual, es importante que si no quieres perder el archivo 'desordenado' realices una copia antes de correr el programa. El siguiente archivo el de iteraciones, se creará en la carpeta 'Iteraciones_Radix' la cual si no existe el mismo programa la crea. Dentro de esta carpeta se irán guardando los archivos de las iteraciones de los archivos que se ordenen por Radix. El nombre de estos será ' Iter_Radix_[Nombre Archivo]'.

Una vez habiendo aclarado los archivos, el ordenamiento se llevaba a cabo gracias a tres métodos: radix, clasificarMiembro y regresarArchivo. El primero radix lo que hace es ir leyendo el archivo original linea por linea y le va pasando cada linea al método clasificarMiembro. Éste lo que hace es separar la linea en nombre, apellidos y numero de cuenta usando la función String.split(","). Esto nos hace ya poder trabajar con el numero de cuenta. Según la iteración el dígito significativo que se analizará. Conociendo el valor del dígito se envía la línea completa al archivo auxiliar cola. Una vez se hayan clasificado todas las linea gracias a un ciclo en el método radix, se llama al método regresarArchivo el cual se encarga de descargar los archivos auxiliares en el archivo original que se quería ordenar y en el archivo de registro de iteraciones. En el caso del archivo original, este se borra antes de que se le vacíen los archivos auxiliares. El de iteraciones lo añade al final para mantener un registro. Finalmente esto se repite el numero de veces del largo del numero de cuenta que es la clave por la que se está ordenando(en este caso son claves de 6 digitos).

De esta manera es que funciona a grandes rasgos el algoritmo Radix externo. Una última cosa importante de mencionar es que el archivo de iteraciones también se va a sobre escribir si se realiza Radix Sort para dos archivos con el mismo nombre (o el mismo archivo dos veces). Por lo que si se quiere mantener el registro de dos ordenamientos del mismo archivo, se deberán o renombrar manualmente los archivos dentro de la carpeta 'Iteraciones Radix' o copiarlos en otro lado antes de hacer nuevamente el algoritmo a un archivo del mismo nombre.

Conclusiones

David Calderón

Este proyecto fue algo que no era tan fácil de trabajar, al principio pensé ´´Porque hacer proyecto en un lenguaje de POO si con programación estructurada nos hubiéramos tardado la mitad de este tiempo´´ con forme pasaron los días y las lineas de código crecían cada vez más, me pude dar cuenta que el hacer un proyecto no solo es buscar la solución mas fácil pues si este fuera el caso el progreso de la programación se estancaría pues nadie buscaría más haya de lo que resulta sencillo, con forme empecé a adentrarme y conocer más nuestro código entendí que cada quien busca solucionar un problema de maneras distintas esto generaba una discusión la cual daba como resultado una mezcla de todas estas ideas para resolver ese problema.Al final lo mas importante fue la discusión y la comunicación pues a raíz de esto creo que llegamos a una versión del programa la cual para todos era la mas óptima para resolver los 3 problemas que nos propusieron.

Con esto puedo decir que lo más importante es conocer todas las herramientas que tenemos a nuestro alcance pues hasta ahora pudimos resolver este problema con conceptos vistos en Eda 1, Eda 2 y POO, lo cual quiere decir que mientras mas avance la carrera tendremos que cambiar la idea de ´´usare lo aprendido en los últimos meses para resolver un problema´´ a ´´De todo lo que se hasta el momento que puedo utilizar para resolver este problema´´ con esto abriremos muchos camines los cuales pocas personas conocen.

Finalmente puedo decir que estoy satisfecho con los resultados que he tenido con mi equipo pero esto no quiere decir que esto sea lo mejor que se pueda ofrecer, solo queda seguir mejorando tanto individualmente como en grupo.

Humberto Hernández

Este proyecto definitivamente te pone a pensar, porque ya no es como una práctica de laboratorio en donde te dan un procedimiento a seguir, en esta ocasión, tuvimos que organizarnos y crear nuestro propio procedimiento. A pesar de que los algoritmos los conocíamos e hicimos ejercicios en clase cuando los vimos, al momento de implementarlos es cuando te pones a reflexionar, cómo son los datos de entrada, cómo debes tratarlos, qué consideraciones se toman en cuenta para la comparación, ordenamiento, intercalación y escritura en los archivos, incluso el formato para poder facilitar las siguientes iteraciones, de tal forma que se puede ”generalizarün poco el algoritmo, aunque hay algunas líneas que sería lo que se llama *código espagueti*.

Después de varias reescrituras de algoritmos, considero que llegamos a la mejor versión posible que podemos crear en este momento con todo lo que hemos aprendido. Seguramente conforme avancemos en el curso podremos ver de nuevo este proyecto y encontrar formas en las que se optimizaría el código.

Cada etapa de la creación del programa tomó una considerable cantidad de tiempo, primero para resolver cómo hacer el ordenamiento, y de ahí reutilizarlo para la otra variante, y luego checar los archivos que sí fueran escritos en el orden correcto, y que tuvieran la información, en ocasiones se tenían registros de más o de menos, entonces *debuggear* el programa para poder encontrar en donde teníamos esas fugas o inflado de información y poder parcharlas de tal forma que tuviéramos correctamente la información.

Iñaky Ordiales

Los proyectos de programación siempre suelen ser un desafío para nostros los alumnos. Si bien son cosas claramente a nuestro alcance, la verdad es que son mucho más completos y por ende más complejos que a lo que estamos acostumbrados en la práctica de laboratorio. Aunque para se justos de igual manera se nos da mucho más tiempo, además de un equipo para su desarrollo. Por lo que de cierta manera se empieza a equilibrar la cantidad de trabajo y esfuerzo que requiere de nosotros.

El objetivo del proyecto era que aprendiéramos sobre la implementación de los algoritmos de ordenamiento externos, al igual que sobre el manejo de archivos en el lenguaje de programación que eligiéramos. Además del desarrollar habilidades de trabajo en equipo que son necesarias para cualquier desarrollo grande de la vida real. Durante la creación de este programa y con el resultado final se puede concluir que se logró cumplir con los objetivos satisfactoriamente. Al trabajar en equipo nuestra organización fue dividirnos cada uno un algoritmo de ordenamiento y que como fuéramos pudiendo alguien desarrollara el método main desde donde se ejecutarían los algoritmos y se definiría la estructura del programa. Al cada quien trabajar en su algoritmo y manejar los archivos como necesitara correspondientemente, cada uno tuvimos que enfrentarnos a las dificultades de los archivos. Sin embargo, esto no quiere decir que no nos ayudáramos, por el contrario para aprender a manejar los archivos se realizó una investigación individual, pero luego hablamos y explicamos lo que habíamos encontrado. Además de que cada vez que alguno encontraba un método de utilidad para el manejo de archivos o de Strings como lo fue el método split(), nos lo compartíamos.

En este proyecto por lo menos yo sentí que trabajamos bien en equipo ya que cuando uno necesitaba ayuda los otros lo ayudábamos. Esto nos hacía estar familiarizado con el desarrollo del compañero para poder ayudar sin necesidad de ponernos a estudiar y entender todo el código en ese momento. Además cada vez que uno acababa una parte del desarrollo, lo mandaba al grupo de mensajería que tenemos y los otros dos lo revisaban y probaban para buscar posibles errores. Así mis compañeros varias veces me avisaron de algo que no funcionaba del todo bien. Finalmente el haber logrado hacer el programa completamente funcional (con las limitaciones indicadas en el manual de usuario) habla de nuestra comprensión de los algoritmos de ordenamiento, siendo el objetivo principal del proyecto. Hacer una simulación de ordenamiento externo, algo que no se vio en clase. Creo que el resultado es bueno cumpliendo con los requerimientos y siendo un producto final con el que quedo feliz.

Fuentes de información

[1] Charles E. Cormen Thomas H; Leiserson. *Introduction to algorithms*. MIT Press, 2003. ISBN: 0-262-03293-7.

[2] Wayne Pollock. *Creating and Using Java Doc Comments*. URL: `https://wpollock.com/Java/JavaDoc.pdf` (visitado 13-11-2020).

[3] Andy Wicks. *Java - BubleSort*. URL: `https://www.youtube.com/watch?v=RqfWvIsYmsc&ab_channel=AndyWicks` (visitado 09-11-2020).