Proyecto #1: Ordenamiento externo
Estructura de datos y algoritmos II

Argüello Leon, Dante Moisés Gaytán Nava, Aarón Emmanuel
Sánchez Pérez, Marco Antonio

11 de noviembre de 2020

1. 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.

2. Antecedentes

Para poder llevar a cabo el proyecto, fue necesario emplear una serie de conceptos previos muy importantes tanto en la parte conceptual como en la parte práctica. Estos antecedentes son pilares de los procedimientos e implementaciones realizados, por lo que se mencionarán y explicarán a continuación.

2.1. Estructuras de datos lineales

Las estructuras de datos lineales están compuestas por una secuencia de cero o más elementos de algún tipo determinado. Cada componente tiene un único sucesor y predecesor con excepción del primero y último, respectivamente. Estas estructuras permiten operaciones como la inserción y la eliminación de elementos sin alterar su orden interno.

2.2. Lista

La lista es una colección de datos o elementos del mismo tipo, en el que se pueden llevar a cabo operaciones de eliminación, búsqueda e inserción, esto por medio del uso de índices. Esta estructura es una lineal y dinámica

de datos. Lineal porque a cada elemento le puede seguir sólo otro elemento; dinámica porque se puede manejar la memoria de manera flexible, sin necesidad de reservar espacio con antelación. La principal ventaja de manejar un tipo dinámico de datos es que se pueden adquirir posiciones de memoria a medida que se necesitan; éstas se liberan cuando ya no se requieren. El dinamismo de estas estructuras soluciona el problema de decidir cuál es la cantidad óptima de memoria que se debe reservar para un problema específico.

2.3. Listas ligadas

Las listas logadas son colecciones de elementos llamados nodos; el orden entre éstos se establece por medio de un tipo de datos denominados punteros, apuntadores, direcciones o referencias a otros nodos. Como ya se mencionó, las operaciones más importantes que se realizan en las estructuras de datos son las de búsqueda, inserción, y eliminación. Se utilizan también para comparar la eficiencia de las estructuras de datos y de esta forma observar cuál es la estructura que mejor se adapta al tipo de problema que se quiera resolver. La búsqueda, por ejemplo, es una operación que no se puede realizar en forma eficiente en las listas. Por otra parte, las operaciones de inserción y eliminación se efectúan de manera eficiente en este tipo de estructuras de datos.

2.4. Listas simplemente ligadas

Esta estructura fue la que se implementó en el proyecto en gran medida. Una lista simplemente ligada constituye una colección de elementos llamados nodos. El orden entre éstos se establece por medio de punteros o referencias, es decir, direcciones a otros nodos. El nodo consta de dos partes en general: un campo información que será del tipo de los datos que se quiera almacenar en la lista; un campo liga, de tipo puntero, que se utiliza para establecer la liga o el enlace con otro nodo de la lista. Si el nodo fuera el último de la lista, este campo tendrá como valor nulo. Al emplearse el campo liga para relacionar dos nodos, no será necesario almacenar físicamente a los nodos en espacios contiguos.

El apuntador o referencia al inicio de la lista es importante porque permite posicionarnos en el primer nodo de esta, y tener acceso al resto de elementos. Si se llegara a perder esa referencia, entonces se perdería toda la información almacenada en la lista. Las principales operaciones con listas ligadas son las siguientes: recorrido de la lista, inserción de un elemento,

borrado de un elemento y búsqueda de un elemento.

3. Marco Teórico

3.1. Ordenamiento

Ordenar significa reagrupar o reorganizar un conjunto de datos en una secuencia específica. Los procesos de ordenación y búsqueda son frecuentes en muchos ámbitos. La ordenación es una actividad fundamental y relevante en la vida. Y significa permutar elementos de tal forma que queden de acuerdo con una distribución preestablecida. Siempre se sigue alguno de los siguientes criterios: ascendente y descendente. En el procesamiento de daros, a los métodos de ordenamiento se les clasifica en dos grandes categorías, según en donde hayan sido almacenados. La primera categoría se denomina ordenamiento interno, ya que los elementos o componentes de las estructuras se encuentran en memoria principal de la computadora. La segunda categoría se llama ordenamiento externo, ya que los elementos se encuentran en archivos almacenados en dispositivos de almacenamiento secundario, como discos, cintas, tambores, etcétera.

3.2. Ordenamiento interno

Los métodos de ordenamiento interno se pueden aplicar a diferentes tipos de dato, no solo a números o tipos primitivos de los lenguajes de programación. En esencia, los métodos se pueden clasificar en directos y logarítmicos. Los primeros tienen la característica de que su implementación es relativamente sencilla y son fáciles de comprender, aunque son ineficientes cuando el número de elementos de la estructura es de tamaño muy grande. Los métodos logarítmicos, por su parte, son más complejos que los directos. Su elaboración es más sofisticada y, al ser menos intuitivos, resultan ser más difíciles de entender. Sin embargo, son más eficientes ya que requieren de menos comparaciones, movimientos, inserciones u otras operaciones fundamentales para ordenar sus elementos.

Una buena medida de eficiencia entre los distintos métodos la constituye el tiempo de ejecución del algoritmo y éste depende fundamentalmente del número de comparaciones y movimientos que se realicen entre sus elementos. Por lo tanto, se puede concluir que los algoritmos directos se pueden utilizar para entradas pequeñas de datos, mientras que los algoritmos logarítmicos, pueden y deben de usarse en entradas grandes de datos.

Existen más categorías al momento de diferenciar a los métodos de ordenamiento, y estas son las siguientes: por inserción (insertionSort), en donde se toma a un elemento y este se inserta en el lado izquierdo del arreglo que ya se encuentra ordenado, el proceso empieza a partir de la segunda casilla y se aplica hasta el último elemento; por selección (selectionSort), en este lo que se hace es seleccionar el más pequeño y guardarlo en la primera posición, luego el siguiente más pequeño y guardarlo en la segunda posición y así sucesivamente hasta el penúltimo elemento (el ultimo ya no requiere ordenarse); por intercambio (quickSort, y bubbleSort), los cuales se caracterizan porque se intercambian los valores como resultado de la comparación de los mismos, dentro de esta categoría se encuentra uno de los algoritmos más rápidos en cuanto al proceso de ordenamiento, el cual es quickSort; por intercalación (mergeSort), en este tipo se usa la recursividad y en cada recursión se toma una estructura de elementos desordenados, se lo divide en dos mitades, se aplica la recursión en cada una de estas y luego, dado que al finalizar estas recursiones tenemos las dos mitades ordenadas, se intercalan ambas para obtener la estructura ordenada. Existen algunos otros tipos de algoritmos de ordenamiento, los cuales difieren en sus características de desempeño, tiempo, y memoria utilizada. Pero principalmente se encuentran estos.

3.3. Ordenamiento externo

Con anterioridad, fue muy común procesar grandes volúmenes de información fuera de la memoria principal de la computadora, puesto que esta no contaba con el almacenamiento necesario para poder trabajar con tanto. Estos datos se almacenaban en archivos, los cuales se encontraban en dispositivos de almacenamiento secundario, como cintas, discos, etcétera. El proceso de ordenar los datos almacenados en varios archivos se conoce como fusión o mezcla; se entiende por este concepto a la combinación o intercalación de dos o más secuencias ordenadas en una única secuencia ordenada. Se debe de hacer hincapié en que sólo se colocan en la memoria principal de la computadora los datos que se pueden acceder en forma directa.

3.4. Intercalacion de archivos

Por intercalación de archivos se entiende la unión o fusión de datos o más archivos ordenados de acuerdo con un determinado campo clave, en un solo archivo.

3.5. Ordenación de archivos

La ordenación de archivos se efectúa cuando el volumen de datos es demasiado grande y éstos no caben en la memoria principal de la computadora. Al ocurrir esta situación no se pueden aplicar los métodos de ordenación interna antes mencionados, de modo que se debe pensar otro tipo de algoritmos para ordenar datos almacenados en archivos.

Por ordenación de archivos se entiende, entonces, la ordenación o clasificación de éstos, ascendente o descendente, de acuerdo con un campo determinado al que se denominará campo clave. La principal desventaja de esta ordenación es el tiempo de ejecución, debido a las sucesivas operaciones de lectura y escritura al y del archivo. Los dos métodos de ordenamiento externo más importantes son los basados en la mezcla directa y en la mezcla equilibrada.

3.6. Ordenamiento por mezcla directa

El método de ordenamiento por mezcla directa es de los más utilizados en esta categoría de algoritmos. La idea central de este algoritmo consiste en la realización sucesiva de una partición y una fusión que produce secuencias ordenadas de longitud cada vez mayor. En la primera pasada, la partición es de longitud 1 y la fusión o mezcla produce secuencias de longitud dos. En la segunda pasada, la partición es de longitud dos y la fusión produce secuencias ordenadas de longitud cuatro. Este proceso se repite hasta que la longitud de la secuencia para la partición sea la mitad de los elementos más uno.

3.7. Ordenamineto por mezcla equilibrada

El método de ordenación por mezcla equilibrada, conocido también como mezcla natural, es una optimización del método de mezcla directa. La idea central de este algoritmo consiste en realizar las particiones tomando secuencias ordenadas de máxima longitud en lugar de secuencias ordenadas de tamaño fijo previamente determinadas. Luego se realiza la fusión o mezcla de las secuencias ordenadas, en forma alternada sobre dos archivos. Aplicando estas acciones en forma repetida se logrará que el archivo original quede ordenado. Para la realización de este proceso de ordenación se necesitarán tres archivos. El archivo original y dos archivos auxiliares para manejar entradas y salidas, de forma alternada, con el objeto de realizar la fusión y partición. El proceso termina cuando en la realización de una fusión y/o partición el segundo archivo quede vacío, o cuando se detecta

que la máxima longitud de la partición es igual que el tamaño del archivo original.

3.8. Radix Sort

El algoritmo de ordenamiento RadixSort no necesita realizar ninguna comparación entre las claves, tampoco realiza intercambios. Cada uno de los elementos se verifica por separado, y en cada uno de ellos se revisan los valores absolutos de los dígitos o caracteres que conforman una clave. En una implementación común se utilizan estructuras FIFO tales como colas para cada uno de los dígitos en el rango de los dígitos que conforman a las claves. Y su estrategia general es la siguiente:

- Elegir el dígito menos significativo de las claves.
- Recorrer los elementos de la colección.
- Insertar un queue auxiliar adicional correspondiente al dígito revisado.
- Hacer push de cada uno de los dígitos revisados en su respectiva queue.
- Extraer los elementos de las queues y colocarlos de nuevo en la colección.
- Repetir los pasos anteriores con el siguiente dígito menos significativo.

Este algoritmo aprovecha las propiedades de las estructuras FIFO para realizar el ordenamiento, pues podemos observar que el número de iteraciones será definido por la cantidad de dígitos de las claves. Al ordenar claves numéricas, empieza a extraer el dígito menos significativo de cada una de las claves y las va colocando en queues, cuando termina la iteración vacía todas las queues manteniendo un orden ascendente en la correspondencia de los dígitos, de esta forma al finalizar una iteración se tiene en un orden relativo la posición de cada una de las claves, y el orden será mayor mientras el dígito analizado sea más significativo.

4. Análisis

4.1. Main

Para la elaboración de este menú principal del programa, se consideró el uso de paquetes e importación de clases. Esto con el fin de brindar un mayor

orden y permitir una mejor legibilidad del programa. Esta clase importa las clases principales usadas en el programa como las clases encargadas del ordenamiento, la que se encarga del manejo de archivos, o las principales utilerías del lenguaje de programación Java. Este menú tiene como finalidad que el usuario pueda tener la oportunidad de escoger el archivo que se va a ordenar, el algoritmo de ordenamiento externo a usar y el criterio de ordenamiento. Esto esta implementado por medio de diferentes métodos dentro de la clase Main, por medio del uso de ciclos de repetición while junto con estructuras de selección switch-case. En cada método hay instrucciones que sirven para representar submenús diferentes con sus respectivas opciones a escoger en cada caso. Dependiendo del menú escogido, se podrán realizar determinadas acciones.

- Método principal main(String[ ] args): Entonces, dentro del método main, se cuenta con el menú principal para poder escoger entre la selección del archivo con las claves a ordenar, y la opción de salir del programa. Si se selecciona la opción de escoger un archivo, se despliegan los diferentes archivos disponibles para ordenar; así el usuario podrá ingresar, por medio de una cadena, la dirección de ese archivo. El programa brindará indicaciones de cómo es que se debe de ingresar la dirección de ese archivo.

- Método arlgoritmoDeOrdenamiento(String fileName): Una vez seleccionado el archivo, se despliega el submenú de algoritmos de ordenamiento, en donde podremos escoger el algoritmo a usar para ordenar las claves del archivo. Este submenú está implementado en un método fuera del método principal, y se invoca simplemente por medio de una llamada a dicho proceso, pasándose como parámetro la cadena contenedora de la dirección del archivo seleccionado.

Cuando se selecciona el algoritmo a emplear, aparece un nuevo submenú correspondiente a cada algoritmo específico, en el que se le pide al usuario que seleccione alguno de los tres criterios para el ordenamiento. Al terminarse el proceso de ordenamiento de las claves contenidas en el archivo seleccionado, en pantalla aparece un mensaje mostrando la localización del archivo final ordenado junto con los archivos auxiliares empleados con las iteraciones del proceso realizado, según sea el caso. Después de llevarse a cabo el ordenamiento, el submenú del algoritmo elegido aparece nuevamente y se le da la oportunidad al usuario de ordenar nuevamente ese archivo, pero por medio de un criterio diferente. En el caso de que se seleccione dicha opción, se realizará el procedimiento por el nuevo criterio. En el caso de que

se quiera salir de ese submenú, el programa permite retornar al submenú de algoritmos y tratar de utilizar uno diferente con el mismo archivo. De esta manera, podemos movernos entre algoritmos de ordenamiento y entre criterios para manejar las claves del archivo seleccionado a conveniencia. Si se quiere usar otro archivo, simplemente se selecciona la opción de salir hasta que se llegue al menú principal y desde ahí se puede elegir el archivo a utilizar. Este fue el análisis de los elementos, métodos y características del menú principal para el funcionamiento e interacción entre el usuario y el programa.

4.2. Paquete Polyphase

4.2.1. Clase PolyphaseSort

Esta clase es la encargada de realizar el ordenamiento por polifase, lo primero que hace es crear un objeto de tipo FileManager y crear los archivos correspondientes a los archivos auxiliares (3), seguido de esto crea un archivo llamado fileIT que es donde se verán reflejadas todas las iteraciones de nuestro algoritmo. Seguido de esto empieza a leer los bloques de tamaño n indicado en el parámetro del constructor del objeto.

Después de esto realiza el ordenamiento del primer bloque de elementos, dicho ordenamiento utiliza un algoritmo de ordenamiento interno, para nuestro caso es QuickSort. Para el ordenamiento interno se utiliza una clase llamada KeyUtilities en el paquete de Utilities:

- KeysUtilities: Es una clase que realiza el ordenamiento interno de la clase su único método es sortBlock(ListString keys, String fileName), el cual recibe la lista de claves a ordenar y el nombre del archivo auxiliar en el que se escribirán, dentro de éste método las claves se ordenan utilizando la clase QuickSort.

Después de terminar este primer ordenamiento se procede a leer nuevamente los bloques creados en los archivos auxiliares y fusionarlos a través de un método de intercalación implementado en la clase KeysIntercalation en el paquete de Utilities:

- KeysIntercalation: Es la clase que realiza la intercalación de las llaves contenidas en dos archivos para ello usa dos métodos el primero y más importante es intercalation(ListString keyOne, ListString keyTwo, String fileName), sus parámetros keyOne y keyTwo son las dos listas que contienen las claves a intercalar dichas listas fueron creadas previamente con la lectura de los dos archivos auxiliares, el parámetro

fileName corresponde al nombre del archivo donde vamos a ingresar nuestro nuevo bloque de claves ordenado, lo único que hace éste método es ir comparando el primer elemento de amas listas e introducir en una nueva lista el elemento de menor valor lexicográfico, esto hasta que las dos listas queden vacías y la tercer lista contenga las claves ordenadas, por último imprime dicha lista final gracias al método printArray(String fileName, ListString keys), el cual recibe dicha lista y la imprime clave por clave en el archivo auxiliar ingresado en el parámetro fileName.

Dicho proceso de lectura e intercalación se realiza hasta que el tamaño del bloque sea mayor o igual al numero de claves leídas al principio. Cumplido este caso significa que nuestro archivo ha quedado ordenado. Es importante mencionar que para elegir la parte de nuestra String a ordenar se usa la clase StringUtilities contenida en nuestro paquete Utilities, la cual convierte nuestra clave completa en nombre, apellido o número de cuenta para su comparación. Al finalizar el ordenamiento nuestro programa nos indica donde esta ahora nuestro archivo con las claves ordenadas y en caso de que haya cambiado de nuestro archivo original deberá salir y elegir el nuevo archivo que contiene las claves, además todas las iteraciones se guardan en fileIT.

4.3. Paquete Mezcla

4.3.1. Clase Mezcla – Mezcla Equilibrada

Para implementar este algoritmo de ordenamiento externo, se hizo uso de clases auxiliares diseñadas por los miembros del equipo, las cuales pertenecen a paquetes diferentes del programa. Esas clases se utilizaron principalmente para el manejo de archivos, mientras que, en la clase contenedora del algoritmo, se escribieron las instrucciones para los tres diferentes tipos de ordenamiento solicitados. Estos tipos fueron para el ordenamiento por nombres, por apellidos y por números de cuenta, de esta manera, se aplica un criterio diferente al momento de solicitar el reacomodo de los elementos.

Debido a que en el menú principal se le solicita al usuario que ingrese el nombre del archivo a ordenar, el método implementado para el algoritmo de mezcla natural recibe como parámetro, en cadena, la dirección relativa del archivo dentro del programa. Cabe mencionar que, los métodos para la ejecución de mezcla natural se establecieron como estáticos, esto es porque es recomendable colocar como estáticos a los procedimientos de propósito general, además no se necesitan instanciar objetos para usar dicho procedimiento. Retomando el método, este recibe la cadena de la localización en

memoria del archivo con las claves a ordenar. Con esa cadena junto con dos cadenas más, ya establecidas dentro del programa, se crean tres archivos. Dos de esos archivos sirven como los dos archivos auxiliares que necesita mezcla natural para almacenar las particiones de tamaño máximo del archivo original, y para mostrar las iteraciones del proceso de ordenamiento. En el tercer archivo creado se muestran las iteraciones del procedimiento, pero mostrando las combinaciones de las particiones que se encuentran en los dos archivos auxiliares. Es preciso mencionar que este tercer archivo solo tiene como finalidad mostrar las iteraciones de las combinaciones entre los dos auxiliares, es decir, se pudo implementar correctamente el algoritmo solo con esos dos archivos y mostrar las iteraciones en el archivo original, pero se quiso hacer de la manera antes mencionada porque así el archivo original permanece con las claves iniciales ordenadas. De esta manera, el archivo original se puede volver a utilizar para realizar el ordenamiento por medio de otro criterio o incluso por otro algoritmo. Por lo tanto, dos de los archivos sirvieron como auxiliares para el funcionamiento principal del algoritmo de mezcla natural, mientras que el tercer archivo sirve simplemente como evidencia de las iteraciones llevadas a cabo en la combinación de las claves de los dos archivos auxiliares.

Como la implementación del algoritmo se trata de una simulación, una vez leídas las claves del archivo de extensión txt, se declaran listas ligadas para el tratamiento de las claves dentro del programa, es decir, se utilizan estructuras y colecciones de java como las listas ligadas para trabajar con los datos leídos del archivo. Entonces, se declaran tres listas, una para fungir como la colección original, la cual tendrá todos los datos leídos del archivo ingresado, y las otras dos para las colecciones auxiliares propias del algoritmo. Así mismo, se establecieron algunas variables auxiliares enteras y booleanas para poder tener el manejo de las posiciones, los índices, y las intercalaciones dentro del programa.

Una vez declaradas e inicializadas las variables y estructuras, tenemos un ciclo de repetición do-while, el cual tiene el control completo del algoritmo, pues se ejecutará las veces necesarias hasta que la colección con las claves se encuentre ordenada totalmente. Entonces, dentro de este ciclo de repetición, se tiene otro ciclo, pero esta vez es un while que tiene como propósito revisar qué elementos están ordenados de forma natural, para así ingresarlos de forma intercalada en cada lista auxiliar. Con esto, se reparten las particiones en bloques de máxima longitud con las claves de manera intercalada entre las dos listas auxiliares. Para lograrlo, se compara a cada elemento con el anterior, pero verificando que se encuentren ordenados, si se llega al punto en que dos elementos ya no están ordenados de forma natural,

se toma hasta ese punto la partición de tamaño máximo y se ingresan los elementos de esa partición a una lista auxiliar (también se van añadiendo las particiones y elementos a los archivos), después se realiza la misma comparación con los siguientes elementos y la nueva partición se ingresará en la otra lista auxiliar con ayuda de una bandera booleana que cambiará de estado intermitentemente.

Después de realizar ese proceso (ciclo while), ya se tienen las particiones de tamaño máximo en las dos listas auxiliares, así como en los archivos. A continuación, se tienen las instrucciones para recuperar las particiones de las estructuras auxiliares y poder mezclarlas de forma ordenada dentro de la colección original. Al realizar este ordenamiento de la mezcla entre las particiones de los dos archivos auxiliares, se toma en cuenta la longitud de esta nueva partición o bloque producido por la unión, para no tomar u ordenar elementos fuera de los límites de ese bloque. Para lograrlo, se usó un ciclo que se ejecuta hasta que detecte que las listas auxiliares están vacías, lo que indica que la lista original ya tiene a los bloques de mayor longitud ordenados. Dentro de este ciclo, se extraen las particiones de máxima longitud de cada lista auxiliar y se insertan en la lista original, posteriormente se realiza el ordenamiento por medio de una implementación de insertionSort que sólo ordena a los elementos comprendidos en ese bloque, es decir, se respeta el accionar de mezcla natural porque solo se ordenan las claves que se encuentran en ese nuevo bloque generado por la unión de las dos particiones. Al terminar el procedimiento de este ciclo de repetición, las listas auxiliares quedan vacías, y la lista original cuenta con los bloques combinados ordenados. Debo mencionar que en los archivos se van escribiendo las iteraciones, mostrándose los bloques separados por espacios e indicándose el número de iteración.

Hasta este punto termina el procedimiento del ciclo de repetición principal, el cual ejecutará todo lo anteriormente descrito, hasta que se detecte que la lista original tiene a todos los elementos ordenados. Esta verificación se realiza dentro del mismo ciclo, pues en el momento en que se compara a elemento por elemento para comprobar que se encuentren ordenados de forma natural, se tiene un contador que indica el número de elementos ordenados; si este número de elementos ordenados es igual al tamaño de la lista original en algún punto de la ejecución, entonces la lista se encuentra ordenada. De esta manera se terminaría la ejecución del algoritmo y las claves se encontrarían en los archivos.

Debido a que las claves del archivo deben ser ordenadas por diferentes criterios, en esta clase contenedora de mezcla equilibrada, se escribió el algoritmo tres veces, y así se tuvo una versión para ordenar por nombres, por

apellidos y por número de cuenta. En esencia, el procedimiento es el mismo, solo hay diferencias en cuanto a las instrucciones que se encargan de hacer las comparaciones. Esto es evidente si comparamos a la versión del algoritmo que ordena por nombre y a la versión que ordena por apellidos. Puede que suene que es el mismo procedimiento para estas dos versiones por manejar los métodos de los Strings, pero no es así puesto que, para comparar a los elementos, se necesitan usar métodos y variables auxiliares para tener el control de las apariciones de las comas que se encuentran en medio de las claves. Con eso quiero decir que surgió una dificultad más al tratar de realizar las comparaciones y el ordenamiento en general utilizando las cadenas, caracteres y números que se encontraron en medio de las claves, pues se tuvieron que tomar en cuenta los índices internos de las cadenas para comparar a los apellidos, por ejemplo. En el caso del ordenamiento por número de cuenta, debo mencionar que este no tan fue difícil una vez realizado la implementación por apellidos, pues en este punto ya había resuelto el problema de los índices internos de las cadenas con los cuales poder comparar apropiadamente subcadenas de las claves. Finalmente, al finalizar cada versión de mezcla natural, en los archivos se mostrarán los bloques, las combinaciones y las iteraciones del funcionamiento. Este fue el análisis de la implementación del algoritmo de mezcla equilibrada para simular el ordenamiento externo utilizando claves recuperadas de archivos.

4.4. Paquete RadixSort

4.4.1. Clase RadixSortNumeros

El paquete RadixSort contiene a la clase RadixSortNumeros, que a la vez incluye al método Radix. El método Radix ordena las claves contenidas en un archivo por medio del método RadixSort, y recibe como parámetro la dirección del archivo con las claves, este archivo es ingresado por el usuario en la clase Main. La clase RadixSortNumeros tiene como atributos una LinkedList para almacenar las claves leídas en distintos puntos de la ejecución, también tenemos composición de clases con una instancia de la clase FileManager, útil para poder manejar adecuadamente los archivos durante las iteraciones. Se tiene una variable que almacena el conteo de las claves que se están ordenando, dos String auxiliares para manejar individualmente cada una de las claves y tres variables enteras que funcionan como iteradores en los distintos ciclos for contenidos en este programa.

Dado que los números de cuenta de los alumnos constan de 6 dígitos, tendremos 6 iteraciones para cualquier ejecución de este algoritmo, y como

los números de cuenta de los alumnos tienen dígitos en un rango [0,9] se crea antes de entrar a las iteraciones, 54 archivos auxiliares para cada uno de los 9 dígitos en cada una de las 6 iteraciones, de esta manera cumplimos con una parte fundamental del ordenamiento externo por medio de Radix, ya que cada una de las claves tendrá su respectivo archivo. Aunque debemos tener en cuenta que, si existe alguna colisión, los archivos serán compartidos. Ya que estamos organizando archivos, hacemos uso de ellos de manera implícita como si fueran estructuras FIFO, en vez de usar colas, usamos a los mismos archivos. Para comprender esto, explicaremos más a detalle el funcionamiento de cada una de las iteraciones de este algoritmo:

Antes de empezar con las iteraciones, leemos todas las claves contenidas en el archivo ingresado por el usuario y las colocamos en una LinkedList, para tener un acceso más práctico en las operaciones. Para cada uno de los dígitos, empezando desde el menos significativo, y para cada uno de los nueve archivos correspondientes a cada uno de los dígitos en el rango [0,9] empezando desde el cero, así para cada una de las claves empezando desde la primera leída hasta finalizar las contenidas en la lista que las almacena, comparamos el dígito menos significativo actual con el índice del ciclo que controla a los archivos, para identificar si escribimos o no la clave en dicho archivo. Dado que la escritura se da de manera secuencial, en la lectura de esos archivos siempre el primero que se lea, será el primero que haya sido ingresado, siendo esto una estructura FIFO implícita.

Al terminar con la escritura de las claves en sus respectivos archivos, se necesita actualizar la lista que contiene las claves para la siguiente iteración. Así que se vacía la lista y se lee cada uno de los archivos correspondientes a cada una de las claves, en el orden ascendente 0 a 9 según corresponda, y se van añadiendo las claves contenidas en dichos archivos sobre la lista, recordemos que siempre se lee la primera clave ingresada (FIFO). Para obtener el resumen de extracción de claves, se hace un nuevo archivo con las claves y su orden relativo en la lista para cada una de las iteraciones, y continuamos con la siguiente iteración utilizando la "nueva" lista. Al finalizar la sexta iteración, tenemos las claves ordenadas en un archivo, los resúmenes de extracción de claves y los archivos que almacenaron las claves.

4.5. Paquete FileManager

4.5.1. Clase FileManager

Para el manejo de todos los archivos se creó la clase FileManager, la cual posee distintos métodos para poder crear archivos, leer los archivos y

escribir dentro de los archivos.

- Método readBlockFile(String fileName): Éste método nos permite leer el archivo que contiene a nuestras claves o a los auxiliares por bloques, el bloque queda definido por el atributo "n" de nuestro objeto creado, este atributo es el que deberá ser modificado para poder cambiar el tamaño de los bloques a extraer, el parámetro fileName corresponde al nombre del archivo donde se encuentran las llaves. Este método utiliza el atributo de pivote para saber si ha terminado de leer los archivos y/o saber en que elemento del archivo se quedó leyendo. Regresa una lista de tipo String que contiene todas las claves leídas de ese bloque.

- Método readKeyFile(String fileName): Éste método lee completamente un archivo que contiene a nuestras claves o a los auxiliares, el parámetro fileName corresponde al nombre del archivo donde se encuentran las llaves. Regresa una lista de tipo String que contiene todas las claves leídas.

- Método countKeySize(String fileName): Éste método cuenta el número de claves contenidas en nuestro archivo, el parámetro fileName corresponde al nombre del archivo donde se encuentran las llaves. Regresa un número entero que corresponde al número de claves contenidas en nuestro archivo.

- Método writeKeyFile(String fileName, String keyToWrite, boolean reWrite): Éste método es el encargado de escribir o sobrescribir en el archivo deseado, el parámetro fileName corresponde al nombre del archivo donde se desea escribir, el parámetro keyToWrite corresponde a la String o clave que deseamos escribir en el archivo y el parámetro reWrite nos indica si se va a sobrescribir el archivo o la cadena se va a escribir al final de nuestro archivo.

- Método createFile(String fileName): Crea un archivo, el parámetro fileName corresponde al nombre del archivo que se desea crear.

4.6. Paquete OrdenamientoInterno

4.6.1. Clase MergeSortNumeros

Esta clase se encuentra en el paquete OrdenamientoInterno, consiste en tres métodos, el método merge, el método sort y el método printArray. Y es funcional para realizar el ordenamiento de los números de cuenta.

a) sort()

Recibe un arreglo, su índice principal y su índice final, se encarga de verificar el caso base (es recursivo) donde una lista sea de un elemento o esté vacía, si aún no se llega al caso base divide la lista en dos partes: izquierda y derecha; para cada parte vuelve a llamarse recursivamente.

b) merge()

Una vez que la lista esté dividida totalmente, se trabajará sobre el arreglo original para realizar los acomodos (mezclas) de los elementos individuales. Internamente crea dos arreglos auxiliares, de igual manera uno para la parte izquierda y otro para la parte derecha (determinados en la función sort). Para mezclarlos las comparaciones empiezan par a par con el primer elemento de cada división, no necesariamente mantienen una naturaleza par a par en el transcurso, pero siempre se recorren las divisiones hacia la derecha, aumentando en uno. Se determina cuál elemento de qué división es menor, para colocarlo en el arreglo original, y por cada iteración una variable "k" nos ayuda a determinar dónde fue la última posición que se escribió, esto por si al terminar de comparar sobraron elementos de alguna división, y colocarlos a partir del último elemento sobrescrito del arreglo original.

4.6.2. Clase QuickSortStrings

La clase QuickSortString pertenece al paquete OrdenamientoInterno, y es implementada para ordenar los apellidos paternos de los alumnos. Representa al algoritmo de ordenamiento QuickSort que es un algoritmo basado en la téctica divide y vencerás, cuya estrategia consiste en:

- Elegir un pivote de la lista de elementos.
- Colocar todos los elementos menores a la izquierda y los mayores a la derecha.
- Repetir el proceso de forma recursiva con las sublistaslistas a la izquierda y a la derecha del pivote.
- El caso base son listas vacías o de un solo elemento.

Para los fines de este proyecto, la lista de elementos es la lista de claves contenidas en algún punto de los algoritmos de mezcla, y se accede únicamente a los apellidos de los nombres de los alumnos por medio de el método stringLastName de la clase Utilities a la hora de realizar las comparaciones y determinación del pivote. Nos auxiliamos del método compareTo para hacer la comparación entre Strings, ya que permite un orden alfabético.

5. Conclusiones individuales

5.1. Argüello León, Dante Moisés

Este proyecto fue bastante fructífero en cuanto a conocimiento y modelos prácticos, ya que implementamos correctamente cada uno de los algoritmos de ordenamiento externo en las simulaciones que planteamos, y aunque no ordenamos explícitamente archivos, sí pudimos manipular claves contenidas en archivos y crear archivos auxiliares siguiendo las estrategias de cada uno de los tres algoritmos, y fue bastante agradable ver lo que somos capaces de hacer, puesto que en clase habíamos hecho pruebas de escritorio de las mismas simulaciones, pero saber programarlas conlleva grandes beneficios. Hubo una constante comunicación en mi equipo, lo que facilitó el modularidad de las clases, e implementamos varios de los conceptos vistos tanto en EDA 2 como en POO, ya que hicimos uso tanto de la organización de clases, así como diversas composiciones entre ellas, y dejamos un programa bien estructurado. Comprendimos cosas fundamentales del manejo de archivos, tanto para escritura como para lectura y esto era una parte del objetivo, por lo que se cumplió. Tuvimos un buen trabajo en equipo, nuestro programa es funcional, y manejamos archivos, fue una buena manera de desarrollar nuestras habilidades de programación. Pero debo mencionar que una de las desventajas que tuve fue no poder implementar Radix para ordenar Strings, ya que se hubieran generado muchos archivos auxiliares, sin embargo, para ordenar los números de cuenta no hubo mayor problema.

5.2. Gaytán Nava, Aarón Emmanuel

Creo que este proyecto fue realmente una prueba para mí, más allá de comprender el funcionamiento del ordenamiento externo un proyecto de este tamaño me enfrentó a lo que muy probablemente vaya a hacer en mi día a día como programador, entender los algoritmos no es difícil, realizar e implementar cualquiera de los ordenamientos externos no es el reto completo, creo que el verdadero reto que afronté en esta práctica fue manejar mi desesperación cuando algo no me salía, tener paciencia para usar la depuración e investigar en internet como se manejaban los archivos, todas estas cosas que llevan un tiempo necesitan que yo sepa administrarme y guardar lugar para poder realizar un proyecto decente que al final para mi significa lidiar con nuevos problemas, sé que mi conclusión puede parecer más personal que relacionada con la materia pero esta vez el proyecto me llevó a un límite en el que puso a prueba mi inteligencia emocional más allá de la que poseo en programación, le agradezco si ha leído hasta aquí y no ha pensado en bajar-

me la calificación pero debo mencionarle eso primero porque fue lo que más me dejó más allá de la programación, creo que esta vez digo sinceramente que se cumplieron más allá de los objetivos del proyecto, debido a que el funcionamiento de los algoritmos de ordenamiento externo quedó perfectamente claro desde que se vio en clase, los demás recursos que necesité para la implementación de cada uno de los algoritmos lo encontré en internet, o lo aprendí en mi clase de POO, pero no se enseña en las clases de programación a trabajar bajo presión, a administrar el tiempo y a acoplarse con sus compañeros de equipo, los cuales me demostraron un increíble trabajo en equipo, creo que el proyecto se llevó a cabo gracias a esto último ya que todos contribuimos un pedacito en cada uno de los algoritmos y aunque algunas veces no nos entendíamos siempre mantuvimos comunicación para saber qué hacía x o y método. Al final me llevo mucho más que los ordenamientos externos, el manejo de archivos o esas cosas que se aprenden en la teoría, me llevo la práctica de realizar un proyecto grande a nivel universidad.

5.3. Sánchez Pérez, Marco Antonio

Después de haber realizado el proyecto número uno de la asignatura de Estructura de datos y algoritmos II, puedo concluir que se lograron realizar, de buena manera, las implementaciones de los algoritmos de ordenamiento externo, esto con el fin de conocer y asimilar completamente las propiedades y procesos propios de la lógica algorítmica, intrínsecos en el desarrollo de programas tomando en cuenta la gran importancia y relevancia del ordenamiento en múltiples aplicaciones de optimización. Así mismo, se lograron comprender, captar y aplicar satisfactoriamente las características, la sintaxis, los procedimientos y conocimientos fundamentales relacionados con el manejo de archivos en un lenguaje de programación. En cuanto a esto, debo mencionar que es de suma importancia, ya que constantemente se debe de establecer comunicación con la computadora para el manejo de información entre esta y el usuario, por lo que el manejo de archivos fue indispensable de conocer y utilizar. Siguiendo esta misma línea, se tuvo la oportunidad de aplicar y reforzar los conocimientos adquiridos sobre programación general, así como el uso primordial del lenguaje de programación orientado a objetos Java. Considero que esto fue tan importante como la implementación propia de los algoritmos, pues al aplicar conceptos de un paradigma diferente al revisado en semestres anteriores, tuvimos la oportunidad de desarrollar nuevas habilidades y capacidades, así mismo se nos brindaron más herramientas, áreas y visiones diferentes sobre cómo implementar programas para resolver problemas desde diferentes perspectivas. Finalmente, sin

restarle importancia, debo mencionar que se fomentó, desarrolló y llevó a cabo el trabajo en equipo de manera significativa a lo largo de la elaboración del proyecto. Esto se dio así debido a que los miembros del equipo mantuvimos comunicación constante a lo largo de las semanas, por medio de reuniones periódicas en plataformas virtuales como Google Meet, así como en plataformas de mensajería instantánea. En dichas reuniones se compartían opiniones, ideas y estrategias para la correcta implementación de los algoritmos, archivos, y los conceptos aprendidos sobre programación en general. Con esto logramos desarrollar competencias sociales indispensables para nuestro desenvolvimiento en la vida laboral y personal, las llamadas habilidades blandas.

6. Conclusión General

Consideramos que el objetivo de este proyecto se cumplió, debido a que pudimos implementar cabalmente los tres algoritmos de ordenamiento externo sobre claves contenidas en archivos, pudimos mostrar las iteraciones en archivos auxiliares y diseñamos una buena estructuración interna del programa. Como equipo nos acoplamos bien a trabajar ya que mantuvimos una comunicación constante por medios digitales, y nos contactábamos siempre de que algún método o clase pudiese interferir en el diseño de aguna parte del programa. Aprendimos a usar los archivos, esto fue una de las partes que más se nos complicó en el diseño de los algoritmos debido a que anteriormente no habíamos trabajado con ellos en el lenguaje Java, pero fué una grata expereicnia ver nuestros resultados. Este proyecto fomentó mucho la investigación entre todos los integrantes, ya que necesitábamos presentar buenos antecedentes, análisis y marco teórico; también fomentó el uso de plataformas digitales, y de entornos exclusivos para programadores como GitHub. Nos agradó la calidad y la dedicación que requería esta práctica, ya que no fue muy fácil programar la teoría y las pruebas de escritorio que habíamos hecho en clase, dado que este proyecto trató con los puntos más finos del ordenamiento externo. No dudamos que la experiencia obtenida en este proyecto nos será de mucha utilidad en el campo laboral, tanto en situaciones de trabajo en equipo como para ordenar archivos.

Referencias

[1] Cairo Battistutti, O., & Guardati Buemo, S. (2006). Estructuras de datos (3a ed.). McGraw-Hill.

http://search.ebscohost.com/login.aspx?direct=true&db=cat02025a&AN=lib.MX001001277430&lang=es&site=eds-live

[2] Guardati Buemo, S. (2007). Estructura de datos orientada a objetos: algoritmos con C++ (Primera edición). Pearson Educación. http://search.ebscohost.com/login.aspx?direct=true&db=cat02025a&AN=lib.MX001001976395&lang=es&site=eds-live

[3] Joyanes Aguilar, L. (2008). Fundamentos de programación: algoritmos, estructura de datos y objetos (Cuarta edición). McGraw-Hill Interamericana. http://search.ebscohost.com/login.aspx?direct=true&db=cat02025a&AN=lib.MX001001698916&lang=es&site=eds-live