Proyecto #1: Métodos de ordenamiento externo
Estructura de datos y algoritmos II

Cabello, Sofía López, Ricardo Núñez, Diego

15 de noviembre de 2020

1. Polifase

Polifase es un algoritmo de ordenamiento externo que requiere de un ordenamiento interno y de 3 archivos auxiliares. El algoritmo básicamente consiste separar la colección inicial de datos en pequeños grupos ordenados, y después, a través de merge, unirlos en uno solo.

Es necesario un algoritmo de ordenamiento interno para ordenar los pequeños grupos de de datos. Los grupos de datos deben caber en memoria interna por esta razón, así que el tamaño de los grupos se suele elegir en función del tamaño del la memoria interna.

El proceso de generación de estos bloques es simple. Primero, se lee el grupo de datos del archivo de entrada(F0) y se ordenan los datos con un algoritmo de ordenamiento interno. Para este proyecto se eligió Insertion Sort, porque el tamaño de los bloques se fijó en 10 datos y porque es un algoritmo sencillo y eficiente para cantidades de datos pequeñas. Después, el grupo ordenado se escribe en un archivo auxiliar (F1) y se repite el proceso para el siguiente grupo, solo que en esta ocasión se escribe en otro archivo auxiliar (F2).

Después de tener los pequeños grupos ordenados, se hace merge entre pares de ellos para generar uno más grande. Este proceso se repite hasta que quede solo un grupo con todos los datos ordenados.

Para hacer el merge, los archivos F1 y F2 se convierten en los archivos de lectura. Luego, el primer bloque de F1 se intercala con el primer bloque de F2 y se escribe el bloque resultante en F0. Luego, se repite el proceso con el segundo bloque de cada archivo, pero esta vez el resultado se escribe en F3. Cuando se terminan los bloques de F1 y F2, F0 y F3 se vuelven los archivos de lectura y F1 y F2 en los de salida. Se repite el proceso hasta que todos los datos estén en un solo archivo.

2. Clase File

Todos los sistemas operativos requieren de pathnames para nombrar archivos y directorios. Esta clase es una vista abstracta e independiente del sistema operativo de los pathnames. Los pathnames consisten en un prefijo específico del sistema y una secuencia de cero o más directorios separados por un carácter también determinado por el sistema(En sistemas unix-like "/ " y en Windows "\"). Además, los pathnames pueden ser absolutos o relativos. Los absolutos son las rutas completas desde la raíz del sistema y los relativos son solo parte de ellas. File por defecto resuelve las rutas relativas al directorio en el que se invocó a la JVM.

Las instancias de esta clase no necesariamente refieren a archivos o directorios que realmente existen. Si representan a un archivo o directorio real, entonces ese archivo o directorio debe existir en el sistema de archivos. Además, las instancia de esta clase no son mutables, por lo que no pueden ser modificadas después de su creación.

El sistema de archivos puede administrar los permisos de archivos y directorios de los cuales esta clase es dependiente. Estos permisos pueden hace que algunos métodos fallen.

Algunos de los métodos que utilizamos en el proyecto son:

- createNewFile(): Crea un nuevo archivo si y solo si el archivo aun no existe.
- delate(): Elimina el archivo si es que existe. Si la instancia refiere a un directorio entonces este debe estar vacío.
- exists(): Verifica que el archivo denotado por el pathname exista.
- getAbsolutePath(): regresa una cadena con la ruta absoluta del archivo.
- isDirectory(): Verifica si la instancia refiere a un directorio.
- listFiles(): Devuelve un arreglo de objetos File de todos los archivos dentro de un directorio.
- mkdir(): Crea un directorio en la ruta de la instancia.

3. Analisis de Polifase

La implementación de polifase se divide en cuatro clases agrupadas en el paquete polifase, y utiliza a las clases dato y alumno del paquete dato. Los

nombres de las clases son Polifase, Merge, InsertionSort y FilesDirect.

3.1. Clase Polifase

La clase polifase es el método principal de polifase. Dentro de ella se pueden encontrar tres variantes del método sort, una para cada criterio de ordenamiento. Lo mismo sucede con las clases Merge e InsertionSort. Para este análisis y explicación del programa, se tomara en cuenta solo la versión para número de cuenta y se mencionarán las diferencias con las variantes.

El método sortNum recibe como parámetro el nombre del archivo que se va a ordenar. Algo importante que notar es que recibe únicamente el nombre del archivo y no su ruta completa, ya que esta se genera más adelante.

Lo primero que hace este método es instanciar algunas de las clases necesarias, como FilesDirect, File, FileReader, BufferedReader, Merge y Dato. Con el método rutaFolder de FilesDirect se obtiene la ruta del directorio donde se guardarán los archivos. En seguida, se crea dicho directorio junto a los archivos.

Luego, el método comienza con el proceso de separar el archivo de entra en bloques. Dentro de un do while, el programa verifica que haya al menos 10 elementos para crear el primer bloque, si no es así, cuenta cuantos elementos realmente hay. Luego, el programa llama al método leer dato de la clase dato. Este método recibe desde que linea debe empezar a leer los datos, hasta que otra linea debe dejar de leerlos y devuelve una lista con objetos Alumno.

Después, esa lista es ordenada con el ordenamiento interno, en este caso con la versión para números de cuenta llamada inSortNum de la clase InsetionSort. Aquí es donde esta la única diferencia entre las diferentes versiones del método polifase, ya que la versión para nombres llama a inSortNom y la versión para apellidos a inSortApe.

posteriormente, se escriben los datos ordenados en el archivo auxiliar F1. Finalmente, el programa verifica que el archivo aun no esté vacío. Si esta vacío, sale del ciclo do-while, si no,vuelve a contar 10 elementos, los ordena y los escribe, esta vez en el archivo F2 (La siguiente escritura será en F1, la siguiente en F2 y así sucesivamente). Este proceso se repite hasta que no haya más datos que leer en el archivo original.

Una vez que terminó de separar los datos del archivo original entre los archivos F1 y F2, el método llama al método MergeNum de la clase Merge.

3.2. Clase Merge

La clase merge tienen tres métodos, mergeNum, mergeApe y mergeNom, cada una para un criterio de ordenamiento. Como ya se dijo, solo se explicará la versión para número de cuenta y se harán notar las pequeñas diferencias entre las variantes.

Cunado se instancía a esta clase, el constructor obtiene la ruta al directorio donde se guardarán los archivos con una instancia de la clase FilesDirect, que se almacena como un atributo del objeto.

Los métodos de Merge reciben 5 parámetros, la ruta de los dos archivos de los que se va a leer, la de los dos archivos a los que se va a escribir y un parámetro para tener un recuento de cuántas llamadas recursivas se han hecho.

Lo primero que se hace es crear una instancia de la clases File, FileReader y BufferedReader para cada archivo de entrada. En la primera iteración los archivos de entrada serán F1 y F2 que son los archivos donde se colocaron los bloque ordenados internamente. Después, con ayuda del parámetro que nos dice en que llamada recursiva se está, se calcula la cantidad de arrobas que se tienen que contar para llegar a la iteración correcta en los archivos. Las diferentes iteraciones de los archivos se separan por arrobas, por esta razón, es necesario hacer lo anterior.

Luego,con lo obtenido, se posiciona a los lectores de ambos archivo en la primera linea de la iteración. En seguida, se le la primera linea; si está vacía, se deduce que la iteración también lo está, lo que significa que el archivo ya está ordenado porque todos los datos ya están en un solo archivo. Si sucede esto, se despliega un mensaje indicando el archivo donde están los datos ordenados y se detienen las llamadas recursivas. Si las primeras lineas si tienen contenido, se procede a combinar los datos de ambos archivos.

En un while que se repite mientras no se llegue al final de archivo, se ejecuta otro while mientras no se llegue al final del archivo o del bloque de datos. El primer while se repite mientras siga habiendo datos en la iteración y el segundo solo mientras haya datos en el bloque de datos actual.

Dentro del segundo while, se convierten las lineas leídas a objetos de tipo Alumno con ayuda del método obtenerDato de la clase dato. Luego, se compara el atributo de número de cuenta de ambos objetos. Los objetos están nombrados como alumL y alumR (izquierda y derecha) donde en el caso de la primera iteración alumR proviene del archivo F1 y alumL de F2. Si el número de cuenta de alumR es mayor al de alumL, se escribe en uno de los archivos, en caso contrario se escribe a alumL. Finalmente, se lee la siguiente linea del archivo cuyo alumno tuvo un número de cuenta menor.

Los archivos de escritura se van intercambiando, es decir, en la primero iteración del while se escribe en F4, la segunda en F3 la tercera en F4 y así sucesivamente hasta que se llegue al final del bloque de datos de uno de los dos archivos.

En las comparaciones anteriores es donde se pueden encontrar las diferencias entre las versiones del método merge. Para nombres, se compara el atributo de nombre de los dos alumnos con el método compareTo justo después de haberles aplicado el método toUpperCase de las cadenas. Si pudo haber utilizado el método compareToIgnoreCase, pero al momento de programar esto no se conocía su existencia. Lo mismo ocurre en la versión para apellidos, pero en lugar de ocupar el atributo de nombre, se utiliza el de apellido.

Con este proceso, se logran unir dos de los bloque de una iteración de los archivos, pero hay un problema. Como en el proceso anterior el ciclo se detiene cuando se alcanzaba un espacio en blanco en solo uno de los archivos, pueden quedar alumnos en el bloque de datos del otro archivo. Por esta razón, en dos ciclos while(Uno para el caso en que los elementos hayan quedado en F1 y otro para el caso en que hayan quedado en F2) se leen y escriben los datos sobrantes. Este proceso se repite para todos los bloques presentes en la iteración.

al final de este ciclo, se lee la siguiente linea de ambos archivos. Si está vacía o es null en alguno de los dos archivos, significa que no hay un bloque más, por lo que se sale del ciclo. Si tienen contenido se repite el ciclo.

Después de terminar el ciclo anterior, se terminaron de unir todos los pares de archivos, pero ¿Qué pasa si alguno de los archivos tiene un bloque más que el otro? Ese último bloque es ignorado. Por esta razón, después de salir del ciclo anterior, otro par de ciclos (Uno para el caso en que el bloque extra hayan quedado en F1 y otro para el caso en que haya quedado en F2) se leen los datos del archivo de entrada y se escriben en el correspondiente de salida.

Luego de terminar el proceso, se hace una llamada recursiva al mismo método, donde los archivos de lectura se vuelven los de escritura y los de escritura los de lectura. Finalmente, se utiliza el método close de las instancias de bufferedReader y FileReader.

3.3. Clase InsertionSort

Como su nombre indica, esta clase almacena métodos para realizar Insertion Sort con los diferentes criterios de ordenamiento. Se explicará la versión para número de cuenta y se expondrán las diferencias entre las versiones.

Este método recibe un ArrayList de objetos Alumno.

Después, con el método size de ArratList se obtiene la cantidad de elemento de la lista. Luego, se recorre la lista desde 1 hasta $n - 1$ y cada elemento se compara con los que están antes de él (que están ordenados) hasta que se encuentre uno que sea menor o se llegue al comienzo de la lista. En ese momento, se inserta al elemento justo después del elemento que es menor que él, o al inicio de la lista, si es que es el menor de todos. Por último, el método devuelve la lista ordenada.

Para hacer las comparaciones entre elementos se utiliza el getter getNoCuenta() de los alumnos y el comparador de relación <= de java. Aquí es donde esta la diferencia entre las versiones. para nombres y apellidos se utiliza el método compareTo de las cadenas. Para evitar errores, se utiliza el método toUpperCase justo antes del anterior.

4. Clase Main

La clase Main contiene al método main del del programa. EL método main sirve únicamente para desplegar el menú y para llamar a los métodos de los diferentes ordenamientos. Primero, pide al usuario el nombre del archivo a ordenar y lo guarda como cadena. Es importante que el nombre del archivo ingresado sea correcto, ya que si no lo es, se producen errores en el programa al momento de tratar de abrir el archivo.

Luego, despliega un menú con los diferentes ordenamientos. Cuando el usuario elige una opción, el programa la almacena como una cadena. Después, en una serie de if-else, se utiliza el método equals de las cadenas para comparar lo ingresado por el usuario con las diferentes opciones. Se eligió esta aproximación debido a que evita que el programa se cierre de manera inesperada por el ingreso de tipos de datos diferentes a los esperados.

Si el usuario elige cualquiera de los algoritmos, se despliega otro menú que le indica elegir por que criterio quiere ordenar los datos. Al igual que antes, se recibe la opción del usuario como cadena para evitar errores relacionados con el tipo de dato. Después, esa cadena se convierte en un entero con el método valueOf y en un switch se valida la opción del usuario.

En caso de quiera ordenar con Mezcla equilibrada, el programa utiliza polimorfismo para ordenar por el criterio correcto. A una variable de la clase Mezcla equilibrada, se le asigna la referencia a una instancia de su clases hijas MezclaNombre, MezclaApellido o MezclaCuenta dependiendo del criterio de ordenamiento elegido. Por último, se llama al método mezcla de la clase padre.

En caso de elegir polifase, se creará una instancia de la clase de mismo nombre. Dependiendo del criterio de ordenamiento, se llamará la al método sortNum, sortNom o sortApe para ordenar por número de cuenta, nombre o apellido respectivamente.

Por último, en caso de elegir radix, se creará una instancia de RadixSort y se llamará a los métodos RadixSortN, RadixSortA RadixC para ordenar por nombre, apellido o numero de cuenta respectivamente, dependiendo de la opción elegida por el usuario.