Proyecto 1. Estructuras de Datos y Algoritmos II

Barrera Peña Víctor Miguel
Roldán Franco Luis Miguel
Velázquez de León Lavarrios Alvar

Entrega 25/03/2019

1. Introducción

Los algoritmos de ordenamiento han sido fundamentales en el campo de la computación debido a que nos permiten manejar con mayor facilidad los datos almacenados en una estructura, optimizan el trabajo de otros algoritmos, además de que son la base para programas que implican un mayor nivel de análisis y de programación. De los algoritmos es importante conocer la eficiencia de los mismos, ya que no todos realizan esta tarea con las mismas características, cada uno implica un costo distinto ya sea en cuanto a la eficiencia en rapidez (¿quién ordena más rápido?), y/o la memoria adicional utilizada.

Todo esto se realiza a nivel de memoria principal, pero cuando se desea ordenar desde memoria externa, es decir archivos, implica otros costos y otra forma de interpretar a los algoritmos. Por lo general, el ordenamiento externo se da cuando se tiene un enorme flujo de información el cual no puede ser procesado por completo desde la memoria principal, sino que se tienen que procesar datos por bloques y guardar los resultados parciales en archivos auxiliares.

En este documento se analizarán algunos de los métodos de ordenamiento externo.

2. Marco Teórico

2.1. Algoritmo de Mezcla equilibrada

Consiste en una mejora del método de ordenamiento por Mezcla Directa. Se trata de ir leyendo los elementos del archivo tomando secuencias que se encuentran ordenadas. Cuando se lee un elemento que ya no está ordenado con los leidos previamente se guardan dicho bloque en un primer archivo auxiliar. Se continúa leyendo el archivo principal hasta que la segunda secuencia también esté ordenada y se guarda el bloque en un segundo archivo auxiliar. Se repite el procedimiento alternando los bloques en ambos archivos hasta que el archivo

de entrada haya sido leído completamente.

La mezcla entre los bloques de ambos archivos también se realiza de manera alternada ordenando los elementos y sobre-escribiendo el archivo original. Se vuelve a leer el mismo archivo y se repiten los procedimientos.

2.2. Algoritmo Radix

Como su nombre lo indica, este algoritmo de ordenamiento está basado en el método de *RadixSort*. El algoritmo lee determinado dígito de cada una de las cadenas y las va almacenando en determinada cola. De las colas se generan los archivos. Hay un archivo para cada uno de los dígitos leidos, y de estos se genera un nuevo archivo con el ordenamiento de los elementos por cierto dígito. El procedimiento se repite hasta que se hayan leído todos los dígitos de cada una de las cadenas.

Radix Sort consiste en tener un conjunto de datos desordenados asociados a una estructura, en este caso un archivo. En donde se determinará el tamaño maximo presente en esta estructura. En base al tamaño se realizará n recorridos que se puede decir que en cada uno de ellos se tomara ira tomando un identificador para ser asignado a una cola. para ellos tomaremos el siguiente algoritmo:

1. Tomar el primer elemento de el archivo origen.
2. seleccionar el dígito de menor valor posicional y en base a ello agregar a el archivo asociado a ese numero. por ejemplo 153. Se iría a el archivo 3.
3. Repetir el punto 2, pero ahora con el siguiente numero del archivo, hasta que se termine los números del archivo origen.
4. Obtener los datos del archivo $0, 1, ..., 9$ en ese orden, al obtener los datos de cada archivo añadirlos a el archivo origen.
5. Repetir Desde el punto 1 pero ahora tomando el siguiente valor posicional, hasta el ultimo valor posicional, por ejemplo en el caso del 153 , el ultimo valor posicional es 1.

2.3. Algortimo de Polifase

El ordenamiento por el método de polifase consiste en mezclar los elementos de un archivo hasta que esté vacío. La ordenamiento se da de la siguiente manera:

1. Lee m llaves, se ordenan mediante un método interno y se genera un primer bloque, el cual se guarda en un primer archivo auxiliar.
2. Lee otras m llaves, se ordenan y se genera otro bloque que se guarda en un segundo archivo auxiliar.

3. Se leen otras m llaves y se ordenan, pero se guardan en el primer archivo en un segundo bloque. Se repiten los pasos anteriores.

La mezcla de los bloques se realiza de la siguiente manera:

1. Se intercalan los primeros bloques de cada archivo auxiliar, se ordenan y se guardan en un tercer archivo auxiliar.
2. Los siguientes bloques se ordenan y guardan en un cuarto archivo.
3. Se repiten los pasos hasta que se genere un único archivo con todas las claves ordenadas.

3. Antecedentes

Para la implementación de los métodos todos tienen como base fundamental a la clase *BufferedReader* de Java la cual se encarga de leer el contenido de los archivos. Anexada a esta también se encuentran otras clases del paquete *java.io* como *FileInputStream*, *InputStreamReader* y *File*.

Otra clase fundamental para los tres programas fue *PrintWriter* para la sobre-escritura de los archivos.

Para que todas las clases anteriores funcionaran se tuvieron que exportar las ecxepciones *IOException* y *FileNotFoundexception* debido a que durante la ejecución de los programas se podría dar un error en la entrada o no se encontraba el archivo a leer.

Debido a que se están ordenando cadenas de caracteres algunos de los métodos como Polifase requirieron una clase llamada *StringBuilder* para convertir cada uno de los caracteres leídos en un String y guardarlo en una lista para su posterior ordenamiento.

Para el menú principal se utilizó a la clase *Scanner* para leer desde teclado y asignar el valor a alguna de las opciones del menú mediante un switch.

A continuación se muestran algunas fuentes consultadas para la realización del proyecto:

- Class BufferedReader
  `https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`
- External Sorting
  `https://es.slideshare.net/Krish_ver2/39-external-sorting`

- Java.io.BufferedReader.read() Method
  `https://www.tutorialspoint.com/java/io/bufferedreader_read_char_len.htm`
- Class StringBuilder
  `https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html`

4. Análisis y Desarrollo

4.1. Implementación

Para comprender de manera detallada la implementación de los algoritmos se recomienda leer la documentación de cada uno de ellos. A continuación se presenta una descripción general de cada uno de los algoritmos.

Explicación del menu principal El menú principal está compuesto por dos opciones: si se elije 0 terminará la ejecución del programa, si se elige 1 se abrirá un submenú donde se seleccionará un método para ordenar, si se introduce algún otro número el programa entrará en un ciclo hasta que el usuario seleccione correctamente alguna de las dos opciones anteriores.

En el menú de para seleccionar los métodos se encuentran las siguientes opciones: 0 para regresar al menú principal, 1 para el ordenaimento por Distribución (Radix), 2 para el ordenamiento por Mezcla Equilibrada, 3 para el ordenamiento por Polifase, y si se mete algún otro valor se entrará en un ciclo hasta seleccionar alguna de las opciones mencionadas.

Explicación de Polifase El primer paso del programa es determinar la ruta del archivo que se va a leer, para ello se utilizaron las clases de java.io pero a su vez se utilizaron las sentencias *try-catch*, ya que se pueden generar excepciones de entrada/salida durante la ejecución del programa.

Se utilizó un objeto de tipo *StringBuilder* que fue almacenando los caracteres de cada llave. Cuando se encontrase una coma durante la lectura del archivo la clave se genera como un String y se guarda en una lista dependiendo el bloque al que pertenece. La lista se ordena mediante el método *bubbleSort()* y las claves se escriben en su respectivo archivo con la ayuda del método *escribirArchivo()*.

Al asignar bloques con un tamaño fijo por lo general quedan algunas claves sobrantes en el archivo, por lo que también se deben ordenar, se genera un bloque con tamaño distinto y se escribe al archivo al que pertenece. A pesar de haber un bloque de tamaño distinto al solicitado este no afecta a lecturas posteriores.

Luego de generar los archivos auxiliares ambos se leen y las llaves que contienen sus bloques se mezclan y ordenan con el método *intercalarArchivos()* creando así un nuevo bloque con el doble de tamaño. Los nuevos bloques nones y pares generados se escriben en su respectivo archivo (impar o par). Al igual que con la lectura del archivo inicial siempre se generan bloques de tamaño irregular, por lo que se hace el mismo procedimiento para acoplarlos al archivo auxiliar correspondiente.

El paso anterior se repite hasta que todas las claves se encuentren en un solo archivo auxiliar, que por lo general es un archivo impar. Por lo tanto, el proceso termina hasta que el último archivo auxiliar par generado se encuentre vacío.

Explicación Radix Sort El programa lee caracter a caracter, cuando detecte una coma se genera un nuevo elemento y se asigna a un archivo auxiliar con el mismo nombre del dígito actual que se está comparando. Se realiza una mezcla en orden de los archivos auxiliares y se sobre-escribe el archivo original. Se vuelve a leer el archivo de entrada y se repiten los pasos con los siguientes dígitos.

Explicación de Mezcla equilibrada El funcionamiento del programa se basa en leer un archivo dado, caracter a caracter e irlos añadiendo a una cadena que se guarda en una lista. Cuando un elemento es mayor al anterior se imprimen los contenidos de la lista en uno de los archivos agregando ':' como separación entre bloques y la lista se vacía. Después los dos archivos auxiliares se leen y se crean 2 listas cuyos contenidos se irán comparando e imprimiendo en el archivo hasta que alguna de las listas quede vacía, en ese caso se terminara de vaciar la lista restante en el archivo. Este procesos se repetira hasta que todos los bloques hayan sido insertados en el archivo original. Finalmente el proceso se repetira, hasta que quede completamente ordenado.

4.2. Pruebas

Pruebas previas de código En esta sección se adicionan las pruebas previas del código que se fue implementando en el código final.

- `https://github.com/EzioFenix7/Proyect_eda/tree/master/pruebas%20proyecto` Pruebas de Snippets de código del que se basa el proyecto final

Pruebas de la versión final del proyecto En esta sección se muestra el programa funcionando en una computadora con sistema Windows, para probar que el proyecto funcione, no se ha dicho que no funcione en otro sistema operativo, pero para caso de análisis lo hacemos solo en este.

4.3. Documentación

- `https://github.com/EzioFenix7/Proyect_eda/tree/master/Documentacion/Polifase` Direccion de documentacion de polifase
- `https://github.com/EzioFenix7/Proyect_eda/tree/master/Documentacion/Radix` Direccion de documentacion de Radix
- `https://github.com/EzioFenix7/Proyect_eda/tree/master/Documentacion/Equilibrada` Direccion de documentacion de Mezcla Equilbrada

5. Conclusiones

El ordenamiento por Polifase se pudo implementar correctamente debido a que se generaron los archivos con las iteraciones adecuadas, mostrando la posición en la que debería ir cada elemento. Internamente puede llegar a ser ineficiente debido a que las claves leidas las ordena mediante el método de *BubbleSort*, así como también de requerir memoria adicional para almacenar los datos listas temporales y ocupar varios objetos de algunas clases de Java para la lectura/escritura de los archivos. Su ejecución se pudo realizar satisfactoriamente en los sistemas operativos Windows y MacOS.

Mezcla equilibrada es un método bastante rápido pero un poco dificil de implementar ya que suele terminar con bloques de grandes tamaños asi que para el uso de listas podría no ser tan eficiente. Tambien como la lectura con *BufferedReader* regresa un entero este debe convertirse a un caracter para poder formar el elemento a evaluar, adicionalmente este método se realizo en una mac asi que los valores devueltos corresponden a unicode por lo cual se deberan hacer ciertos ajustes para su uso en otros sistemas operativos.