UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Ingeniería en Computación

EDA II

Proyecto

Ordenamiento Externo

Equipo 5

Cervantes Barragán José Ricardo

Mendoza González Mario

Romero Torres Brenda Mónica

1 Objetivos

1.1 Objetivo general

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

1.2 Objetivo del equipo

El equipo implementara algoritmos de ordenamiento externo con el fin de poder ordenar archivos, los cuales contienen Nombre, APaterno y Núm. de cuenta, se aplicaran los conceptos teóricos adquiridos en clases, y la experiencia practica adquirida en los laboratorios, todo esto con el fin de desarrollar un buen trabajo y dar el mejor esfuerzo.

2 Antecedentes

Durante el curso, lo visto en la unidad 1 (algoritmos de ordenamiento), se ha trabajado solo con algoritmos de ordenamiento interno, las practicas de laboratorio realizadas han consistido solo de ordenamiento interno, lo cual sirvió bastante para poder realizar el presente proyecto. Aunque el proyecto no consistió en utilizar los algoritmos de ordenamiento interno de manera explicita, funcionaron para poder realizar el ordenamiento externo, ya que todo ordenamiento externo depende de un algoritmo de ordenamiento interno.

El ordena externo fue visto en clase de manera teórica y ademas se realizaron pruebas de escritorio con algunos ejercicios, por lo tanto, esto de ordenamiento externo no fue algo desconocido para el equipo, cabe resaltar que el implementar el ordenamiento externo fue un reto difícil, claro que no es algo imposible, pero si ameritaba demasiada atención, porque a diferencia del ordenamiento interno, el cual solo se basaba en ordenar arreglos, ahora se pedía el poder ordenar archivos externos, archivos los cuales contenían información variada y se tenia que ordenar bajo el criterio que se quisiera.

El programa que se pedía en el proyecto podía haberse realizado en tres lenguajes de programación diferentes: Java, C, etc, claro que nosotros preferimos trabajar con Java, ya que ha sido el lenguaje que se ha ocupado en el semestre, ademas de que es un lenguaje muy amigable y que con todas las herramientas que ofrece facilita muchas tareas, simplifica muchas instrucciones, hace cosas maravillosas, tal vez hubiera sido mejor trabajar con otro lenguaje y hubiera sido mas fácil, pero en opinión del equipo, me parece que no, así que mejor se trabajo con Java, se hizo uso de los conocimiento de toda la unidad 1, ademas que gracias a que el equipo esta cursando la materia De Programación Orientada a Objetos, fue mas fácil hacer uso de Java, y hacer uso de conocimientos del paradigma orientado a objetos.

El equipo cuenta con una base de conocimientos solida de la materia, y sobre todo se espera poder realizar el mejor esfuerzo en el proyecto.

3 Marco Teórico

Algoritmo

Los algoritmos son una secuencia lógica y detallada de pasos para solucionar un problema.

Algoritmos de ordenamiento

Los algoritmos de ordenamiento consisten en el reacomodo de una colección de elementos de tal forma que éstos cumplan una lógica o un criterio para establecer una secuencia de acuerdo a una regla predefinida.

Existen diversas clasificaciones para los métodos de ordenamiento. La más utilizada es aquella que los jerarquiza de acuerdo a la forma en la que utiliza la memoria en la computadora.

- Ordenamiento interno

  El ordenamiento interno se lleva a cabo completamente en la memoria de acceso aleatorio de alta velocidad de la computadora, es decir, todos los elementos que se ordenan caben en la memoria principal de la computadora.

- Ordenamiento externo

  El ordenamiento externo funciona cuando no cabe toda la información en memoria principal, es decir, es necesario ocupar memoria secundaria.

En el presente trabajo solo se hablara de algoritmos de ordenamiento externo, ya que estos fueron los implementados en el proyecto.

Algoritmos de ordenamiento externo

El procedimiento de ordenar el contenido de un archivo o de un bloque muy grande de información almacenado en memoria secundara se le conoce como ordenamiento externo.

El ordenar un archivo implica tres cuestiones:

- Leer su contenido
- Procesar el contenido
- Reescribir el contenido

La mayoría de los algoritmos de ordenamiento externo utiliza la siguiente estrategia general:

- Dividir el archivo a ordenar, en bloques de tamaño manipulable en memoria principal
- Tratar los bloques individualmente
- Hacer una nueva escritura mediante la mezcla de los bloques ya ordenados

La eficiencia de los ordenamientos externos está determinada por el tiempo de acceso al dispositivo externo.

En el ordenamiento externo puede variar:

- La forma de dividir y de mezclar los bloques
- El uso de archivos auxiliares
- La salida final ordenada

A continuación se dará una breve explicación sobre los algoritmos de ordenamiento externo que se utilizaron en el presente proyecto:

Polifase

Consiste en aplicar una estrategia de generación de bloques mientras se realiza la lectura del archivo original.

Este algoritmo de ordenamiento externo maneja cuatro bloques F_0, F_1, F_2, F_3 como archivos auxiliares para guardar la distribución de los arreglos, y supone que en F_0 se encuentran las llaves originales.

ALGORITMO

Ordenamiento externo de cuatro bloques.

Fase 1. Construcción y distribución de los arreglos ordenados. Mientras existan datos en F_0 , los pasos a seguir son:

1. Lectura de n claves.
2. Ordenar las n claves por algún algoritmo de ordenamiento interno.
3. Generar un bloque con esas claves y colocarlo en un archivo auxiliar F_1.
4. Repetir los pasos 1-3 y colocar el nuevo bloque en un segundo archivo auxiliar F_2.
5. Los siguientes bloques generados se intercalan entre los archivos F_1 y F_2.

Fase 2. Intercalación de los arreglos.

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. Repetir los pasos 1 y 2 hasta que no haya mas claves por procesar.

El algoritmo no contempla la posibilidad de que existan más arreglos en un archivo que en otro, esto se debe a la cantidad de datos y de n. Si esto sucede, el arreglo que no tenga pareja será copiado al bloque de salida correspondiente.

Ejemplo:

F_O = [10, 42, 50, 80, 20, 15, 19, 70, 75, 69, 55, 8, 14, 30]

Consideremos a $n = 4$

F_O = [10, 42, 50, 80, 20, 15, 19, 70, 75, 69, 55, 8, 14, 30]

Los números rojos se colocan en F_1 en arreglos de 4 elementos ordenados

F_1 = [10, 42, 50, 80], [8, 55, 69, 75]

Los números azules se colocan en F_2 en arreglos de 4 elementos ordenados

F_2 = [15, 19, 20, 70], [14, 30]

Intercalación de archivos F_1 y F_2

F_0 = [10, 15, 19, 20, 42, 50, 70, 80]

F_3 = [8, 14, 30, 55, 69, 75]

Intercalación de archivos F_0 y F_3

F_0 = [8, 10, 14, 15, 19, 20, 30, 42, 50, 55, 69, 70, 75, 80] <- Conjunto de elementos ordenados

Mezcla equilibrada

La idea central de este algoritmo consiste en realizar las particiones tomando secuencias ordenadas de máxima longitud en lugar de secuencias de tamaño fijo. Luego se realiza la fusión de las secuencias ordenadas, las secuencias se ordenan y se almacenan de manera alternada sobre dos archivos. Aplicando estas acciones en forma repetida se logrará el archivo original quede ordenado. Para la realización de este proceso de ordenación se necesitaran tres archivos. El archivo original F_0 y dos archivos auxiliares a los que se denominara F_1 y F_2.

ALGORITMO

Fase 1. Lectura y división del archivo original

1. Leer una secuencia ordenada de máxima longitud 2. Generar un bloque con la secuencia ordenada y colocarla en un archivo auxiliar F_1
3. Repetir el paso 1 y 2 y colocar el nuevo bloque en un segundo archivo auxiliar F_2
4. Las siguientes secuencias generadas se intercalan entre los archivos F_1 y F_2.

Fase 2. Ordenamiento y salida

1. Intercalar los bloques de los archivos auxiliares 1 y 2 de forma ordenada y dejar el resultado en el archivo original.
2. Si hay mas de un bloque en el archivo original, almacenar estos de manera alternada en los archivos auxiliares F_1 y F_2, hasta que solo quede un bloque.
2. Repetir el paso 1 y 2 hasta que no haya más bloques que intercalar

Ejemplo:

F_O = [25, 30, 40, 3, 18, 16, 40, 90, 33,66, 50, 1, 20, 29, 49, 99]

Secuencias ordenadas

F_O = [25, 30, 40, 3, 18, 16, 40, 90, 33, 66, 50, 1, 20, 29, 49, 99]

Los números rojos se colocan en F_1

F_1 = [25, 30, 40], [16, 40, 90], [50]

Los números azules se colocan en F_2

F_2 = [3, 18], [33, 66], [1, 20, 29, 49, 99]

Intercalar F_1 y F_2

F_0 = [3, 18, 25, 30, 40], [16, 33, 40, 66, 90], [1, 20, 29, 49, 50, 99]

Como hay más de un bloque en el archivo original, estos bloques se almacenarán de manera alternada en los archivos F_1 y F_2

F_1 = [3, 18, 25, 30, 40], [1, 20, 29, 49, 50, 99]

F_2 = [16, 33, 40, 66, 90]

Intercalar F_1 y F_2

F_0 = [3, 16, 18, 25, 30, 33, 40, 40, 66, 90], [1, 20, 29, 49, 50, 99]

Como sigue habiendo más de un bloque en el archivo original, los bloques se almacenaran de manera alternada en los archivos F_1 y F_2

F_1 = [3, 16, 18, 25, 30, 33, 40, 40, 66, 90]

F_2 = [1, 20, 29, 49, 50, 99]

Ultima intercalación

F_0 = [1, 3, 16, 18, 20, 25, 29, 30, 33, 40, 40, 49, 50, 66, 90, 99] <- Conjunto de elementos ordenados

Radix sort externo

El método de ordenamiento Radix Sort también llamado ordenamiento por residuos puede utilizarse cuando los valores a ordenar están compuestos por secuencias de letras o dígitos que admiten un orden lexicográfico.

El algoritmo ordena utilizando un algoritmo de ordenación estable, las letras o dígitos de forma individual, partiendo desde el que está más a la derecha (menos significativo) y hasta el que se encuentra más a la izquierda (el más significativo).

Para cada elemento a ordenar se efectúan los siguientes pasos:

1. Se insertan en la estructura auxiliar los elementos de la colección, de acuerdo con el dígito que se revisa.
2. Una vez que se revisó toda la colección se retiran los elementos de las estructuras auxiliares.
3. Se repite el proceso para cada posición significativa en referencia a los dígitos (o caracteres) de los elementos de la colección.

Este método externo está basado en el método radix.

La diferencia fundamental radica que en radix interno se utilizan colas para cada elemento de la cadena 0,1,2,3; extrapolando lo anterior, radix externo se utiliza un archivo para cada uno de los símbolos analizados en las cadenas.

Utiliza la filosofía FIFO (First In, First Out).
ALGORITMO

El algoritmo en pseudocódigo del Radix Sort:

RadixSort(A,d) Inicio

Para i=1 hasta i=d

Ordenamiento de A en el digito i Fin

Donde es una lista de elementos, es el número de dígitos o caracteres que tienen los elementos de , si = 1 se refiere al dígito o caracter colocado más a la derecha y cuando = al que está más a la izquierda. El ordenamiento se realiza con algún algoritmo estable.

Ejemplo:

1212, 2111, 1231, 1122, 1211, 3311, 2323, 1133

Se revisa el primer dígito de derecha a izquierda, este es el dígito menos significativo y se coloca en su respectivo archivo

A_1 = 2111, 1231, 1211, 3311

A_2 = 1212, 1122

A_3 = 2323, 1133

Siguiendo con la filosofía FIFO es como se reacomodarán los archivos

2111, 1231, 1211, 3311, 1212, 1122, 2323, 1133

Ahora se revisa el segundo dígito de derecha a izquierda y se coloca en su respectivo archivo

A_1 = 2111, 1211, 3311, 1212

A_2 = 1122, 2323

A_3 = 1231, 1133

Siguiendo con la filosofía FIFO

2111, 1211, 3311, 1212, 1122, 2323, 1231, 1133

Se revisa el tercer dígito de derecha a izquierda

A_1 = 2111, 1122, 1133

A_2 = 1211, 1212, 1231

A_3 = 3311, 2323

Se aplica la filosofía FIFO

2111, 1122, 1133, 1211, 1212, 1231, 3311, 2323

Se revisa el cuarto dígito de derecha a izquierda, este es el dígito mas significativo

A_1 = 1122, 1133, 1211, 1212, 1231

A_2 = 2111, 2323

A_3 = 3311

Y ya por último se reacomodan según la filosofía FIFO (Una vez más)

1122, 1133, 1211, 1212, 1231, 2111, 2323, 3311 <- Elementos ya ordenados

Básicamente esto es una descripción teórica de lo que son los algoritmos de ordenamiento externo, los cuales fueron utilizados en el presente proyecto.

4 Desarrollo y Análisis

Para empezar a desarrollar el programa, se investigó primero la forma en la que se leen archivos de texto en java. Para ello, se necesita hacer uso de la clase File, que permite obtener información sobre archivos y directorios. Se creó una clase llamada Archivo en la que se importó la clase File, que es la representación de un archivo o un el nombre de un directorio donde se encuentra el archivo. Dentro de dicha clase se creó un método leerArchivo donde se requirió del método Scanner para solicitar al usuario el nombre del archivo y guardarlo en variable tipo String para utilizarlo como argumento de la creación de un objeto de tipo File. Aunque es posible ingresar la dirección específica del directorio del archivo, se

decidió que era más sencillo para los usuarios sólo escribir el nombre del archivo con su tipo de archivo, en este caso ".txt", aunque esto signifique que el archivo a leer debe de estar en el mismo directorio con el que estamos trabajando en el momento, es decir, en la carpeta del proyecto de NetBeans.

Una vez que supo la forma en que se creaba un objeto de tipo archivo, lo siguiente fue determinar el modo en el que se acomodarían los datos para mantener continuidad. Teniendo en cuenta que se contaba con tres tipos de información, nombre, apellido y numero de cuenta, no se podía separar en una sola lista de una sola característica para realizar el ordenamiento. La primera opción considerada fue el uso de arreglos bidimensionales, sin embargo, su manejo se complicaría a lo largo de las implementaciones de los algoritmos, así que la opción elegida fue la creación de una lista de listas, en donde cada lista contenida en la lista principal representaría a un alumno, y éste a su vez estaría compuesto con tres listas de Strings que representarían su apellido, nombre y numero de cuenta.

Lo siguiente a hacer fue leer el archivo txt y separarlo por claves mediante las comas contenidas. Para ello se creó un objeto de tipo Scanner que recibía como parámetro el objeto File creado, posteriormente se creó un ciclo while para que se ejecutara siempre que el objeto Scanner tuviese una linea siguiente. Dentro del ciclo se creó un objeto de tipo String que consistía en la línea completa del archivo, conteniendo la información de un alumno. Dicha línea se utilizó como parámetro de otro método scanner para dividir la linea de acuerdo a las comas contenidas que separaban cada tipo de dato del alumno, usando el método .useDelimiter y .next, dichos datos se guardaron mediante el método add en una lista ligada creada que representaba a un individuo. De esta forma, al terminar el ciclo while, se obtiene una lista de listas

Lo siguiente a realizar fueron los menús, que permitieran al usuario interactuar con el programa. El primer menú planeado fue el método menuPrincipal, que forma parte de la clase Archivo y en donde se mostraban las opciones de leer un archivo txt o salir del programa, con la intención de que durante una sola ejecución, se pudiesen leer más de un archivo, sin embargo, al momento de ejecutar el programa, y terminar de utilizar un archivo de texto, aunque el programa solicite el nombre del archivo, y éste se lea (indicado con la impresión de un contador que se añadió en el ciclo while para saber el numero de alumnos de la lista), no se ejecuta el siguiente menú elaborado.

Dicho menú es marte de una clase Utilidades, recibiendo como parámetro la lista de listas creada con el método leerArchivo y despliega las opciones de los algoritmos de ordenamiento disponibles, en este caso Polifase, Mezcla Equilibrada y Radix.

Al seleccionar los métodos por mezcla equilibrada o polifase, se despliega un método subMenu definido en la misma clase Utilidades, para que el usuario determine el criterio a utilizar para el ordenamiento, mientras que para el algoritmo de radix, el criterio de ordenamiento se limita al número de cuenta.

Al seleccionar el método de ordenamiento por polifase, las opciones para seleccionar los criterios de ordenamiento siempre están disponibles, de esta forma, el usuario puede cambiar de criterio sin tener que elegir una y otra vez el método a usar. El submenú recibe la lista de listas como parámetro, así como la opción elegida en el tipo de ordenamiento del metodo menu, esto para usarlo como un condicional para evitar que haya sobrelape de métodos y se haga el ordenamiento de acuerdo al seleccionado por el usuario.

Para el método de ordenamiento por Polifase, se creó una clase llamada Polifase, dentro se creó un método llamado combinar, que recibe la lista de listas creadas y el entero que representa el criterio seleccionado por el usuario para realizar el ordenamiento. Dentro de éste se crean cuatro listas de listas que fingirán como los "archivos" necesarios para ralizar el ordenamiento por Polifase, llamados listaF0, listaF1, listaF2 y lista F3. Dentro también se le da la posibilidad al usuario de que delimite el número de elementos que desea en cada bloque generado en la primera etapa del algoritmo.

Para realizar la división de la lista principal en bloques e intercalarlos en dos listas diferentes, se créo el método dividir. Éste recibe la lista de listas, el criterio de ordenamiento, el número de bloques y las listas de listas F_1 y F_2.

Dentro del método, se declara un contador y un auxiliar inicializados en cero. El contador sirve para recorrer cada uno de los elementos de la lista sin repetir los que ya se tomaron en cuenta. Dentro de un ciclo while en el que el contador debe ser más pequeño que el tamaño de la lista de listas original, se declara otra lista de listas llamada dividida, que almacenará los elementos de la lista original, separándolos por bloques del tamaño indicado por el usuario, mediante un ciclo for, que aumenta el tamaño del contador en cada iteración, en dado caso de que el contador sea igual o mayor que el tamaño de la lista, se añade la lista generada a alguna de las dos listas F_1 o F_2, dependiendo el caso, de esta forma, se generan listas más pequeñas de lo establecido por el usuario en caso de que se hayan sobrado elementos. En caso de que las listas queden con exactamente el numero de elementos deseado, éstas se van guardando de igual manera en las listas F_1 y F_2. Para determinar en qué lista irá cada bloque y asegurar que se guarden de forma intercalada, cada vez que se guarda una lista, el auxiliar aumenta en uno, de esta forma, si el residuo del auxiliar entre 2 es cero, el bloque se guarda en la listaF1, de otra forma, se guarda en la listaF2. Es importante destacar que los bloques

que se van guardando son ordenados previamente por medio de un método ordenar, definido en la misma clase.

Una de las cararísticas del ordenamiento por Polifase, es que requiere un algoritmo de ordenamiento interno para poder realizar el ordenamiento de los bloques, en este caso, se utilizó el algoritmo selection sort, debido a su facilidad de programación. Se creó el método ordenarP, que recibe la lista a ordenar, el criterio de ordenamiento y un auxiliar que servirá para imprimir las listas genereadas en un archivo txt. Al criterio de ordenamiento seleccionado por el usuario se le resta una unidad y se asigna a una nueva variable entera llamada criterio. Se crean dos listas de listas que nos ayudarán a ordenar, sin modificar la lista original, una llamada apellidos y la segunda llamada original. En la primera se copian todos los elementos de la lista recibida como parámetro, mientras que en la segunda, se copia sólo un elemento, que se usará al momento de intercambiar valores durante el algoritmo selection sort. Ya que no se pueden comparar datos de tipo String con los operadores lógicos "$<$" o "$>$", se decidió convertir los datos en cadenas de caracteres, para acceder a los elementos específicos de las listas a utilizar como criterio de ordenamiento, se ocupó el entero criterio, que representa el índice en el que se encuentra dicho valor, en el caso de los apellidos, el índice siempre será cero, en el caso del nombre, el índice será 1 y en el número de cuenta será 2. Para evitar que el programa colapse cuando se encuentren elementos similares, por ejemplo, los apellidos Rodriquez y Rosales, se usó un ciclo while que aumenta el valor de un comparador, que funge como índice, hasta que encuentra elementos diferentes, en el caso del ejemplo, las letras "d" y "s", procediendo a compararlas entre sí. En caso de que el programa encuentre dos coincidencias, por ejemplo, dos apellidos iguales, el comparador se reinicia a cero y se inserta la primera coincidencia encontrada en la lista, sin comparar ningún otro criterio, por lo que el ordenamiento en éste caso no es estable. La lista llamada original, sirve para almacenar el elemento que será reemplazado por el mínimo encontrado, para colocarlo en el índice correspondiente y así evitar la pérdida de información.

Después de cada ordenamiento, se imprime en pantalla la lista ordenada y se llama al método escribir, definido en la misma clase. Éste método escribe la lista ordenada en un archivo txt incluído en la carpeta del proyecto de NetBeans. El método recibe la lista ordenada y un auxiliar, que simplemente sirve para escribir "Archivo F_1" si el residuo del auxiliar entre dos es igual a cero, o "Archivo F_2" en caso de que el número sea impar. Para el método escribir, se volvió a crear una clase File que recibe el nombre del archivo txt. Posteriormente se usó el método constructor FileWriter, que recibe como parámetros el nombre del objeto File y un valor booleano que indica si se pueden almacenar o no los datos conforme se vaya escribiendo sobre él.

Una vez que las listas F_1 y F_2 se llenaban con los bloques ordenados e intercalados de la lista original, se procedía a intercalas los elementos dentro de ellas en las listas F_0 y F_3, para posteriormente volver a intercalarlas en las dos primeras. Para lograr esto, se creó un ciclo do while, con un contador que indicaba el número de iteraciones hechas, para el cao de las iteraciones cero y pares, se llenaban las listas F_0 y F3 con los elementos de las listas F_1 y F_2, mientras que si era impar, se llenaban las listas F_1 y F2 con los elementos de las listas F_0 y F_3.

Para realizar el proceso de intercalación, se creó un método denominado merge, que recibía como parámetros las cuatro listas y un auxiliar con el valor del tamaño de los bloques, para determinar el tamaño de las listas posteriores e ir duplicándolas durante las iteraciones. Para llenar las listas se utilizaron ciclos for que con cada iteración, extraían los elementos de las listas llenas y colocaban elementos en las listas a llenar mediante el método .poll, que extrae el primer elemento de una lista. En caso de que alguna de las dos listas se vaciara, y para evitar valores null, se agregó una condición en la que si alguna de las listas llenas se vaciase, se añadían todos los elementos restantes a la lista F_1 o F_0, según el caso. Y finalmente, en caso de que después de que se llenasen las listas con el tamaño de elementos indicados pero sobraran el las listas, llenas, se añadían todos los elementos sobrantes en la lista F_0 o F_1, también dependiendo del caso.

En cada iteración, los archivos resultantes se van escribiendo en el archivo txt en donde se imprimieron los bloques generados durante la separación de la lista original. Para ello se creó un método escribir archivo, basado en el método escribir, con la diferencia de que éstos consideran cinco posibles valores para el auxiliar, y dependiendo de ésto, escribe si se trata del "archivo" F_0, F_1, F_2 o F_3. Cuando la lista final está ordenada, el valor del auxiliar es 5, lo que indica que la lista se imprimirá en otro archivo txt completamente diferente llamado PolifaseFinal, contenido en la carpeta de NetBeans que contienen el programa.

Una de las desventajas de nuestro método merge es que no puede ordenar los elementos bloque por bloque, ya que la lista que se genera es contínua, por ello, al momento de intercalar, respeta los elementos que deben ir en cada nuevo bloque combinado, pero sin ordenar. Por ello, al terminar el ciclo while del método combinar, se busca la lista no vacía y se acomoda utilizando el método ordenar P.

El ordenamiento por el método de Mezcla Equilibrada, presento muchas dudas sobre como poder implementarlo, ya que ha diferencia del método Polifase, en este método no se toman bloques fijos definidos por el usuario, en Mezcla Equilibrada se tienen que tomar bloques de secuencias ordenadas, lo cual genero mucha confusión sobre como poder realizar esa acción, al principio se consultaron códigos en internet para poder llevarse una idea y tal vez poderla

asentar con los requerimientos de nuestro programa, la consulta de códigos no fue muy útil, ya que no daba una idea clara sobre como poder concretar el programa.

En internet se llegaba a encontrar códigos los cuales cumplían casi con todos los requerimientos del proyecto, pero desafortunadamente es difícil poder entender la idea de alguien mas, son códigos los cuales utilizan herramientas desconocidas y las cuales no son fáciles de entender, por lo cual se decidió solamente recolectar ideas e implementar el código de manera independientemente a los encontrados en internet.

Lo que cabe resalta en todo lo antes mencionado es que el método Mezcla Equilibrada no se pudo implementar, el código se quedo a medias, y finalmente no se puedo concluir, es una decepción para el equipo, porque se trato de entregar un proyecto bien terminado, no se logro, pero cabe destacar que lo mas importante que nos llevamos de esto es el hecho de haber intentado hacerlo.

Para ordenar los datos del Archivo.txt con el algoritmo de RadixSort lo primero que hicimos fue crear una clase con ese nombre. Creamos, además, los archivos planos que RadixSort, en su implementación externa, utiliza como almacenamiento extra. Dentro de la clase RadixSort el primer método que se creo fuel el método sort() donde se planeaba codificar el ordenamiento. Durante la marcha se fueron necesitando más métodos que dentro de la clase RadixSort son las primeras líneas que aparecen, pero que no fueron las primeras en hacerse. Este método recibe en su argumento la lista de listas con los datos del archivo a ordenar (Archivo.txt) generada en la clase Archivo. Ningún método de esta clase regresa nada. Enseguida se enlistan los métodos que se utilizaron para llevar a cabo el ordenamiento y la creación de los archivos con los datos ordenados y las iteraciones realizadas:

- deArchivoAListа():El método lee todas las líneas del archivo y las va almacenando en la lista de listas original. Al final de este proceso elimina este archivo y crea un nuevo archivo en blanco con el mismo nombre. Este proceso simula la extracción de datos de las colas (según el algoritmo original) para almacenar ordenadamente la lista original
- escribirEnArchivos():El método recibe, desde la lista de listas, los datos que tienen un mismo criterio (en este caso un numero), les da formato y los escribe en el Archivo que se le indique.
- Iteraciones():El método escribe, en el Archivo adicional que muestra el procedimiento, el número de una iteración.

- estadoDeLista():El método escribe, en el archivo que se le indique, el estado de la lista de listas en el momento en que se use este método.

- escribirArchivoIteraciones():El método escribe en el archivo adicional, que muestra el procedimiento del ordenamiento, el estado de cada archivo utilizado como almacenamiento externo: El nombre del archivo, y los datos que hay en el.

- escribirArchivOrdenado():El método escribe en el archivo "ArchivoOrdenado.txt" los datos ordenados obtenidos de la ultima iteración en que la lista fue llenada. Les da formato a cada línea separando los datos por comas obteniendo cada dato con la metodología de las colas (FIFO).

- sort():Este método implementa la lógica del algoritmo de RadixSort utilizando los métodos antes explicados para poder lograrlo. El primer paso dentro de este método es reordenar la lista recibida de manera que el primer dato que se muestre al imprimirla sea el número de cuenta y no el nombre; este procedimiento lo realiza por medio de un ciclo for.

  Inmediatamente, crea todos los archivos que se van a utilizar a lo largo de la ejecución (todos los archivos que utiliza el ordenamiento, el archivo ordenado y el archivo con la iteraciones realizadas) y limpia el archivo con las iteraciones (con el objetivo de que el documento quede en blanco si ya se ha ejecutado una vez el programa) por medio de la eliminación de dicho archivo y la creación de otro nuevo pero con el mismo nombre. Este proceso de limpieza también se hace con los archivos que utiliza el ordenamiento (0.txt, 1.txt, 2.txt, 3.txt, etc.) y con el archivo ordenado. En el caso de los archivos del ordenamiento la limpieza se realiza dentro del método deArchivoALista() y en el caso del archivo ordenado se realiza en el método escribirArchivOrdenado().

  Después, por medio de dos ciclo for anidados se recorre cada dígito del dato (en este caso del número de cuenta) y cada elemento de la lista (cada persona). El primer ciclo se encarga de recorrer el número de cuenta y el segundo de recorrer cada persona. Dentro del segundo ciclo se utiliza una estructura de control switch-case para crear cada caso en el que podría entrar el dígito analizado, se crearon 10 casos correspondiente a cada numero natural(0...9) y caso dafoult para cualquier caso que no corresponda a los números naturales. Dentro de cada caso del switch-case se utiliza el método escribirEnArchivos() para escribir el dato que se esta analizando en el archivo que corresponde. Terminado este procedimiento se cierra el primer ciclo for.

En seguida de la estructura de control switch-case, pero dentro del primer ciclo for, continua el método. En este momento se escribe todo el proceso que se ha hecho en el archivo IteracionesRadix.txt por medio del método escribirArchivoIteraciones() para dejar un registro del procedimiento.Primero se escribe la iteración que corresponde con el método Iteraciones() y después los datos que tiene cada archivo extra (los que simulan las colas de RadixSrot) en ese momento con el método escribirArchivoIteraciones() y al último se escribe el estado de la lista de listas en esa iteración.

Terminado el proceso anterior del registro del proceso, se eliminan los datos de la lista de listas con el método clear() de la clase LinkedList con el objetivo de ingresar los datos de los archivos extra de forma ordenada(primero el archivo 0.txt después el 1.txt y hasta llegar al 9.txt) en una lista vacía.

Por último, y para cerrar el segundo ciclo for, se utiliza el método deArchivoAListA() para pasar los datos de cada archivo a la lista de listas que acabamos de vaciar. Cada vez que usamos este método, después de pasar los datos limpia el archivo actual. Terminado este proceso se cierra el segundo ciclo for.

Como salida, el programa imprime para el usuario el nombre del Archivo Ordenado y el nombre del Archivo donde se hizo el registro del proceso.

5 Conclusiones

Durante la realización del proyecto nos vimos con muchas dificultades, sobre todo con el conocimiento de manejos de archivos txt en Java. En general, los menús creados permiten al usuario una fácil interacción y uso de los métodos de ordenamiento, con la capacidad de elegir diferentes criterios de ordenamiento una vez que seleccionó Polifase. Una de las ventajas de nuestro ordenamiento por Polifase, es que el usuario es capaz de determinar el número de elementos que desea por bloque. Otra ventaja es que logra el ordenamiento de las listas con un solo método, independientemente del criterio seleccionado. Lo más complicado en el caso del método de Polifase fue lograr el ordenamiento por bloques, habíamos intentado varias iteraciones dentro del método ordenarP, sin embargo, aunque las sublistas se ordenaban de manera correcta, la formación de los bloques no era la adecuada, por ejemplo, en un primer intendo, los primeros cuatro bloques se intercalaban de manera correcta, pero al momento de incluir los bloques posteriores, éstos sólo se iban acomodando a la lista ordenada, de forma secuencial. Fue gracias a la implementación del método poll que se logró la intercalación

correcta de los bloques, ahora sólo queda lograr que dichos bloques se ordenen conforme se vayan creando, sin afectar a los siguientes elementos de la lista. Otro contratiempo encontrado es que el algoritmo de ordenamiento no ordenaba de manera correcta y no se sabía la razón, hasta que se notó que durante la comparación de elementos, solo se avanzaba una vez en el arreglo de caracteres para comparar el segundo elemento entre sí, de esta forma, si las cadenas de caracteres comparadas tenían más de dos caracteres en común, la comparación no se hacía y el elemento no se intercambiaba, por ello se decidió implementar el ciclo while. Otro problema con el método después de implementar el while fue que si se encontraban dos cadenas de caracteres completamente iguales, por ejemplo, dos "Marías", el programa generaba errores, por lo que se añadió la condición de que si se recorría toda la cadena completa, simplemente el comparador regresara a cero.

En Mezcla Equilibrada se tuvo el problema de que no se pudo implementar el código, fue difícil poder tomar la idea para poder codificar, se hicieron consultas en internet, se vieron videos, que en cierto punto si ayudaron un poco, pero no se pudo concretar una idea solida del método.

Es grande la decepción que se tiene por no poder hacer las cosas, por no poder implementar, codificar cualquier problema, a veces esta decepción es grande que a uno como programador en ocasiones pierde la aspiración de seguir adelante, tal vez esto no es el caso de todos, pero si es frustrante el no poder con algo.

Para RadixSort fueron problemas a la hora de escribir el registro del proceso ya que se debía controlar la escritura de cada iteración. Fuera del método sort() donde se implementaba el ordenamiento de RadixSort, todos los métodos que había en la clase RadixSort.java se utilizaron para poder generar dicho registro. En general en todos los método se utilizaron la excepciones ya que muchas clases del Api de Java te forzaban a utilizar. Tuvimos que aprenderlo por nuestra cuenta ya que en nuestra materia de Programación orientada a objetos aún no nos las habían enseñado.

En general, con todo el trabajo que hicimos creemos que nos llevamos fuertes conocimientos en el manejo de Archivos y Excepciones con java y nos abrió las puertas a más ideas para aplicaciones que podemos crear con estos conocimientos.

Otro problema fue el poder trabajar en conjunto en el programa, dada la situación actual. Lo cuál nos ayudó a buscar herramientas de trabajo a distancia para poder llevarlo a término lo mejor posible, desde Skype y Zoom, hasta plataformas online para generar textos de LaTeX. Ésto nos llevó a darnos cuenta que en la realidad y en la actualidad, la gran mayoría

de programadores y desarrolladores usan esas herramientas y muchas más para trabajar con personas que están incluso del otro lado del mundo, por lo que es necesario que empecemos a familiarizarnos con todas las opciones que tenemos para trabajar a distancia y a aprender a utilizarlas.

6 Referencias

Se hizo uso de las diapositivas de la materia que se encuentran en EDUCAFI, y también del "Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II"

"APUNTES DE ESTRUCTURAS DE DATOS", http://www.ptolomeo.unam.mx

"Análisis, Diseño e Implantación de Algoritmos", http://fcasua.contad.unam.mx