Universidad Nacional Autónoma de México

EDA II Profesor: Tista García Edgar

Realizado por el equipo 4:

Argüello Dante

Gaytán Nava Aarón Emmanuel

Sánchez Pérez Marco Antonio

**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.

* **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.

* **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.

* **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.

* **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.

**Marco Teórico**

* **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.

* **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 así́ 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.

* **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.

* **Intercalación 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.

* **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.

* **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.

* **Ordenamiento por el método de 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.

**Análisis**

* **Menú principal (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.

* **File Manager:** 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.
* **Sort Polyphase**: 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(List<String> 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(List<String> keyOne, List<String> 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, List<String> 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.

* **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.