

FIUBA

75.06 - ORGANIZACIÓN DE DATOS

TRABAJO PRÁCTICO – GRUPO 1

INTEGRANTES:

TAPIA, JIMENA SOLEDAD
FERNÁNDEZ, PAULO MIGUEL
GALLINAL, NICOLÁS ANDRÉS
MARAGGI, SANTIAGO JOSÉ MARÍA

Indice

Introducción.....	3
Manejo de Interfaz.....	4
Obtención de Libro (offset identificador en el archivo de control).....	5
Obtención de Términos.....	7
Triadas y sus Datos de Control.....	8
Consulta de Frases.....	9
Resolución por Proximidad.....	9
Resolución por Relevancia.....	9
Cálculo de Similitud de Documentos.....	10
FrontCoding (Compresión del léxico).....	13
CaseFolding (Normalización de términos).....	13
Hash Expansible en disco.....	14
TABLA DE DIRECCIONES.....	14
BLOQUE.....	14
ARCHIVO DE DATOS.....	14
CUBETA.....	14
EXTENSIÓN DE CUBETAS.....	14
ELEMENTO.....	14
HASH EXTENSIBLE.....	15
EJEMPLOS.....	15
Estructuras de Parseo.....	16
FORMATO DE ARCHIVO DE STOPWORDS.....	16
Casos de Prueba.....	17
Instrucción – Tomar Texto.....	17
Instrucción – Procesar Editorial.....	17
Instrucción – Procesar Autor.....	17
Instrucción – Procesar Título.....	18
Instrucción – Procesar Palabras.....	18
Instrucción – Listar Archivos Tomados.....	18
Instrucción - Obtener Archivo.....	19
Instrucción – Quitar Archivo.....	19
Instrucción – Ver Estructura.....	19
Instrucciones correspondientes a la segunda entrega.....	21
Reporte de BUGS.....	22

Introducción

El desarrollo de este trabajo practico se baso en el diseño de un sistema para resolver el almacenamiento, consulta y obtención de libros de texto.

Se incorporaron practicas de diseño que permitieron modularizar en estructuras desacopladas, que favorezcan el manejo de datos, accesos a disco y administración de recursos.

Para la ejecución de la aplicación se incorporó un archivo de configuración que permite establecer las rutas para los informes, el archivo de control y el que se utilizará como lista de stopwords.

Manejo de Interfaz

Para el manejo de las instrucciones ingresadas por comandos, se diseñó un modelo apto para la adaptación de una posible interfaz gráfica en la que cada comando se separó en un tipo distinto de instrucción.

Cada una de ellas implementa el método *ejecutar()* de manera tal que cumpla con el comportamiento específico del comando ingresado por el usuario.

- Instruccion_ConsultarAutor → `./ejecutable -qa`
- Instruccion_ConsultarTitulo → `./ejecutable -qt`
- Instruccion_ConsultarEditorial → `./ejecutable -qe`
- Instruccion_ConsultarTerminosProximos → `./ejecutable -qp`
- Instruccion_ListarArchivosTomados.h → `./ejecutable -l`
- Instruccion_ObtenerArchivo.h → `./ejecutable -o ID_Archivo`
- Instruccion_ProcesarAutor.h → `./ejecutable -a`
- Instruccion_ProcesarTitulo.h → `./ejecutable -t`
- Instruccion_ProcesarEditorial.h → `./ejecutable -e`
- Instruccion_ProcesarPalabras.h → `./ejecutable -p`
- Instruccion_QuitarArchivo.h → `./ejecutable -q ID_Archivo`
- Instruccion_TomarTexto.h → `./ejecutable -i "archivo de texto"`
- Instruccion_VerEstructura.h → `./ejecutable -v [-e árbol de Editorial, -a árbol de Autor, -t hash de Título, -p hash de Palabra, -at Archivo de Términos, -ani Archivo de Norma Infinito, -aop Archivo de ocurrencia posicional, -li Listas Invertidas] "Nombre Archivo"`

Es importante:

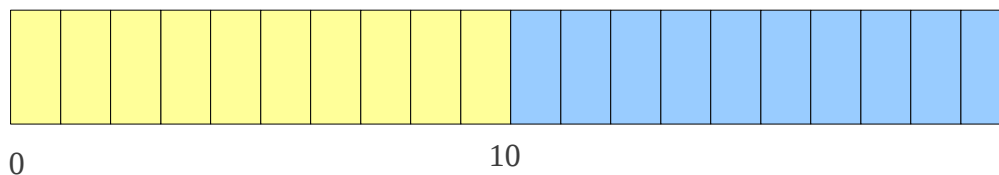
- Tener actualizado el archivo de configuración para que se generen los archivos en el directorio deseado.
- Insertar los parametros que sean strings con comillas dobles ("").

Obtención de Libro (offset identificador en el archivo de control)

El id utilizado para la identificación del registro que se debe recuperar para obtener un libro en particular se corresponde con el offset en el archivo de registros variables.

Suponemos inicialmente un archivo rectangular de 20 particiones equivalentes. En él está guardado un único libro que ocupa 10 particiones, es decir que su offset es 0.

En el caso de existir nuevo libro, se guardará a partir de la última partición ocupada, por lo que su offset de identificación será 10.



Particularmente en nuestro caso, almacenamos una metadata al comienzo de cada libro que indica cual es la longitud del mismo hasta el proximo. De esta manera se recupera del archivo lo necesario, sin tener que recorrerlo todo.

Esta situación en el **archivo de control** se representará de la siguiente manera:

```
0 | 0 | - | - | - | - |
10 | 6 | - | - | - | - |
```

donde el primer número equivale a la primera partición que se tiene que leer del archivo de registros variables, el segundo número al espacio libre hasta el próximo libro y las últimas cuatro a los tipos de indexación posibles (en este caso se suponen no indexados).

Es decir que para el ejemplo, el libro que comienza en la partición 0, tiene de offset 0 y está todo ocupado hasta el próximo libro que comienza en 10. Este segundo libro puede ser de un tamaño máximo de 6, ya que la metadata al comienzo del mismo ocupa lugar.

Se supone ahora que se elimina el primer libro, es decir aquel que comienza en el offset 0 y se vuelve a guardar un libro nuevo en ese espacio libre.

Es ahora que se abren dos posibilidades:

- Que el nuevo libro ocupe el *mismo* espacio que el que estaba guardado anteriormente.
- Que el nuevo libro ocupe *menos* espacio que el que estaba guardado anteriormente.

Nota: no se considera el caso en el que ocupe *mayor* espacio porque de esa manera no se hubiera elegido el offset 0 y se lo estaría colocando más adelante en el archivo de registros variables.

Si se tiene el primer caso, se asigna $idLibro = 0$ y se ocupa todo el espacio disponible hasta el libro contiguo guardado que comienza en el offset 10.

Si se tiene el segundo caso, la asignación sigue siendo $idLibro = 0$ pero para el próximo libro a ser guardado se deberá evaluar la posibilidad de ocupar el espacio que queda libre entre el ocupado por este libro y el ya existente que comienza en el offset 10.

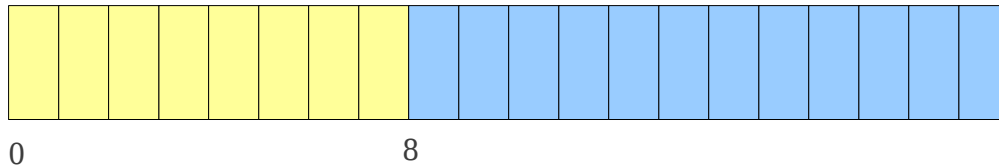
Esta situación en el **archivo de control** se representará de la siguiente manera:

```
0 | 6 | - | - | - | - |
```

10 | 6 | - | - | - | - |

Suponemos que el nuevo libro a guardar tiene un tamaño de 4 particiones. Es decir que quedan libres 2 particiones hasta el próximo offset ocupado.

Si se deseara insertar un nuevo libro, se pueden fusionar los espacios libres. De esta forma, el primer libro arrancaría en cero y se extendería hasta la octava partición inclusive (4 metadata + 4 contenido del libro). Luego un segundo libro podría almacenarse a partir de la octava partición.



Esta situación en el **archivo de control** se representará de la siguiente manera:

0 | 0 | - | - | - | - |
8 | 8 | - | - | - | - |

Para saber si se puede utilizar ese nuevo offset disponible, se debe tener en cuenta el tamaño del nuevo libro a guardar (para el caso debe ser menor o igual a 8 particiones).

Generalizando, el offset que queda disponible para un próximo libro a guardar luego de eliminar uno existente u ocupar parcialmente un espacio, se obtiene como:

$$\text{nuevoOffset} = \text{espacio libre viejo} - \text{id libro recién asignado} + \text{metadata} + \text{tamaño del libro}$$

Y el espacio libre que queda para ese nuevo offset es:

$$\text{nuevoEspacioLibre} = \text{espacio libre viejo} - (\text{próximo id libro} + \text{metadata})$$

Donde la metadata ocupa 4 bytes.

Obtención de Términos

Los términos que se obtienen de cada uno de los libros, se guardan uno a continuación del otro en un archivo de términos.

Dicho archivo se utiliza como índice de términos y sus respectivos identificadores. El id utilizado para la identificación de cada uno de los mismos se obtiene a partir del tamaño del archivo de términos, dado que cada uno de los términos se agrega al final del archivo separado del anterior por el carácter especial “|”.

Los términos están indexados en un árbol el cual mantiene la referencia con las triadas, es decir, el tipo de dato que contiene la relación entre el término, el libro que lo contiene y su posición relativa en dicho libro.

Creación del Índice para RTT:

- 1) guardar el libro => obtenengo "id libro".
- 2) obtener lista de palabras del libro sin stop words.
- 3) pedir al archivo de control de triadas espacio continuo para insertar tantas triadas como palabras contenga el libro.
(Es más sencillo para el proceso de generación de normas y para el proceso de eliminación de libros si se encuentran todas agrupadas).
- 4) para cada palabra (i) del libro:
 - 4.1) verificar si está indexada (hash).
 - 4.1.1) la palabra está indexada, obtengo "id palabra".
 - 4.1.1.1) guardar en el archivo de triadas: <"id libro", "id palabra", posición (i)> => me devuelve el "id triada" (offset en el archivo).
 - 4.1.1.2) guardar en el árbol: <palabra, "id triada">.
 - 4.1.2) la palabra no está indexada.
 - 4.1.2.1) guardar la palabra en el archivo de terminos => me devuelve el "id palabra" (offset en el archivo).
 - 4.1.2.2) guardar en el hash <palabra, "id palabra">.
 - 4.1.2.3) guardar en el archivo de triadas: <"id libro", "id palabra", posición (i)> => me devuelve el "id triada" (offset en el archivo).
 - 4.1.2.4) guardar en el árbol: <palabra, "id triada">.
- 5) guardar en control de triadas el rango en donde está el libro.
- 6) generar normas.

Por ejemplo, los una consulta podría ser:

- 1) obtener por pantalla consulta del usuario.
- 2) para cada término (i) de la consulta.
 - 2.1) buscar en el árbol la lista de "id triada" correspondiente al término (i).
 - 2.1.1) crear objeto Término con nombre = término (i) y lista "id triadas".
- 3) obtener colección de Términos de la consulta.
- 4) obtener los "id libro" pertenecientes a todos los elementos de la colección de Términos.
- 5) resolver consulta por proximidad.
- 6) obtener los "id libro" no pertenecientes a todos los elementos de la colección de Términos.
- 7) resolver consulta por similitud.

Triadas y sus Datos de Control

Las triadas se utilizan para mantener relación entre el término, el libro que lo contiene y su posición relativa en dicho libro.

El formato es del estilo:

< id_Libro , id_Termino , posicionRelativa >

donde:

id_Libro : es el identificador del libro al cual pertenece el término.

id_Termino : es el identificador del término.

posicionRelativa: posicion relativa del término dentro del archivo libro.

Las mismas están controladas por un archivo que contiene datos de control representados por un cuarteto cuyo formato es del estilo:

< id_Libro , id_TriadaInicial , id_TriadaFinal , eliminado >

donde:

id_TriadaInicial : es el identificador de la primer triada que corresponde al archivo libro que se tiene por *id_Libro*.

id_TriadaFinal : es el identificador de la última triada que corresponde al archivo libro que se tiene por *id_Libro*.

eliminado: permite saber si ese libro fue eliminado en algún momento.

El principal motivo de la utilización de una estructura de tríada, es el rápido acceso a datos necesarios para el cálculo de las normas de documento y la resolución de consultas de frases.

De esta manera, cada vez que se consulta un término, se accede mediante el índice de términos, se obtienen las triadas que le corresponden y de ellas la información necesaria para calcular los resultados de la búsqueda.

Consulta de Frases

Cuando el usuario busca una frase en los libros, el sistema otorga dos resultados.

El primero de ellos es por proximidad de términos y el segundo es por la relevancia de los mismos en los libros.

Para realizar esta distinción en los resultados, lo que se hace es lo siguiente:

Se toman todos los términos de la consulta y para cada uno se obtiene la lista de tríadas correspondiente. Luego a este conjunto de términos se le consulta por los libros comunes a todos ellos, estos son los libros que serán listados en el resultado por proximidad.

Los libros restantes, los cuales no contienen todos los términos de la consulta serán listados en el resultado por relevancia.

Resolución por proximidad:

Se realiza mediante un enfoque iterativo, iterando por los términos internos de la consulta.

Para tener una medida de proximidad, cada libro tendrá dos puntajes para dicha consulta, "orden" y "proximidad".

Una vez terminado el proceso cálculo de proximidades se ordenarán los libros primero por "orden" descendente y para igual "orden" por proximidad descendente.

Detalle del proceso de cálculo de proximidades:

1) Obtención de posiciones más próximas de términos en libro:

Para cada término interior de la consulta en dicho libro se le consulta cuáles son las posiciones más próximas de sus hermanos de la consulta en el libro (se guardan dos valores: hermano anterior, hermano posterior). En caso que un hermano no esté por delante o por detrás del término, retorna -1 como posición más próxima.

Este proceso retorna una lista de posiciones más próximas de términos en libro como resultado.

2) Obtención de triada de consulta por proximidad:

Esta triada consta de "id libro", "orden" y "proximidad".

Este proceso toma como input la lista de posiciones más próximas de términos en libro y retorna una triada que representa la resolución de la consulta para dicho libro.

El "orden" se computa sumando la cantidad de posiciones más próximas de términos en libro diferentes de (-1, -1).

La "proximidad" se calcula como la suma de las inversas de las diferencias de posiciones más próximas de términos en libro. Este valor se encuentra entre 0 y 1, donde más cercano a 1 indica una mejor proximidad de los términos de la consulta.

Una vez obtenidas las triadas de consulta por proximidad, se ordenan según lo explicado anteriormente y se listan por pantalla.

Resolución por Relevancia:

Para cada libro y conjuntos de términos se obtiene una medida de similitud utilizando los pesos de los términos y las normas de los documentos.

Luego se ordena según similitud descendente y se listan los resultados.

El proceso se detalla en la sección de cálculo de normas.

Cálculo de Similitud de Documentos

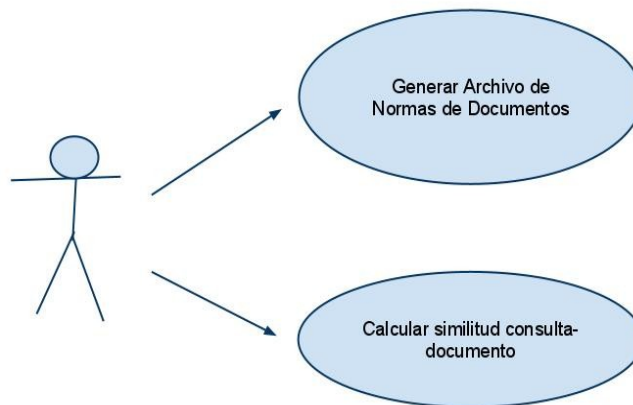
El presente módulo implementa un modelo vectorial para la ponderación de documentos según la colección de términos que los componen. Dicho modelo ofrece la posibilidad de catalogar a los documentos según su “cercanía vectorial”, o similitud. Esta similitud entre dos documentos, o un documento y una consulta, que a nivel del modelo se considera un vector más, se calcula como el coseno vectorial entre ambos.

Casos de Uso

Este módulo presta un servicio, que es el de calcular similitudes entre documentos y consultas. Sin embargo, a nivel funcional, se puede dividir en dos casos de uso: GENERACIÓN DE ARCHIVO DE NORMAS DE DOCUMENTOS, y CÁLCULO DE SIMILITUD consulta-documento.

El archivo de normas de documentos se genera en un hash, el mismo que se desarrolló en la primer etapa de este trabajo práctico. Abajo se ofrece un anexo con un breve instructivo sobre esta estructura, para visualizar su contenido o imprimir su contenido en un archivo (se recomienda ver el ejemplo para interpretar la información binaria en las cubetas del Hash). La estructura usa además otro Hash para indexar los pesos globales de los términos, para mayor eficacia.

Fórmulas de modelo vectorial



$$Pg(x) = \log[10](N / f(x))$$

Donde ...

$Pg(x)$ es el PESO GLOBAL del término x

N es la cantidad de documentos totales contenidos en nuestro sistema

$f(x)$ es la frecuencia global del término, que es la cantidad de documentos que contienen al menos una vez al término x .

$$Pl(x, d) = fl(x, d) * Pg(x)$$

Donde...

$Pl(x)$ es el PESO LOCAL del término x en el documento d

$fl(x, d)$ es la FRECUENCIA LOCAL del término x en el documento d , que es la cantidad de apariciones del término en el documento

$Pg(x)$ es el PESO GLOBAL del término x , arriba descrito

$$v(d) = (Pl(0), Pl(1), \dots, Pl(N))$$

Donde...

$v(d)$ es el VECTOR DEL DOCUMENTO d , y $Pl(i)$ sus pesos locales de término.

$$|d| = (Pl(0)^2 + Pl(1)^2 + \dots + Pl(N)^2)^{(0,5)}$$

Donde...

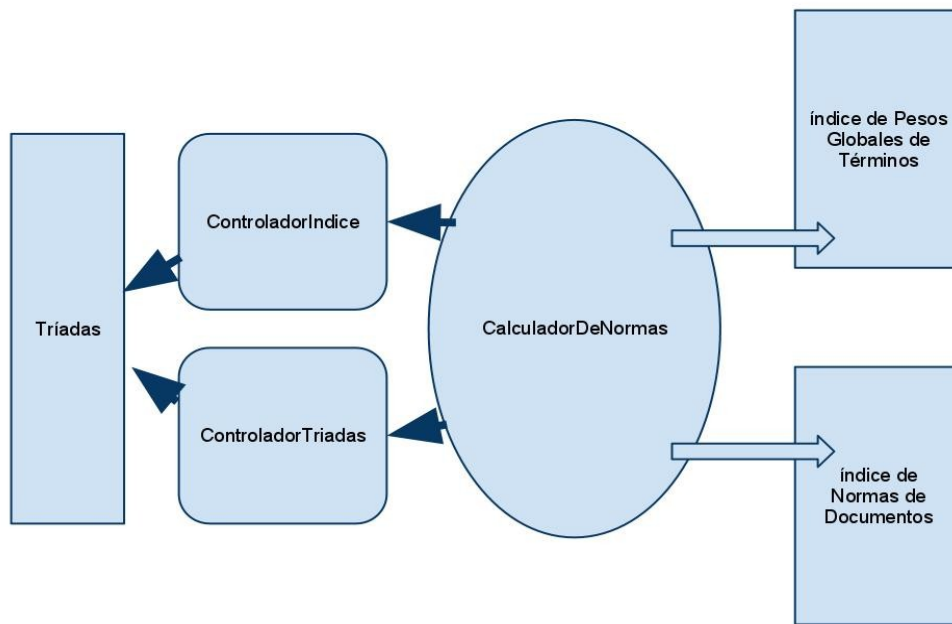
$|d|$ es la NORMA DEL DOCUMENTO, y equivale a la raíz cuadrada de la sumatoria del cuadrado de sus pesos locales.

Estructura dispuesta

Los datos necesarios para los cálculos que implica un modelo vectorial, este módulo los adquiere a través de consultas a la estructura llamada *ControladorIndice* y a otra llamada *ArchivoTerminos*. A pesar del nombre, la última clase es también un controlador del archivo que almacena los términos.

A través del módulo *ControladorIndice* accede en forma indexada a las tríadas, u ocurrencias de términos. Cada elemento tríada guarda en forma unívoca el ID de documento de la ocurrencia, el ID del término ocurrido, y la posición en el documento (que a este modelo no le interesa). Al controlador mencionado se le piden las tríadas pertenecientes a un ID de término determinado, o todas las tríadas referentes a un documento. Para el primer caso se recurre a un índice de tipo ARBOL B+, que opera en disco, desarrollado en la primer etapa del presente trabajo práctico. Para obtener todas las ocurrencias de un documento dado, se recurre al *ControladorTriadas*.

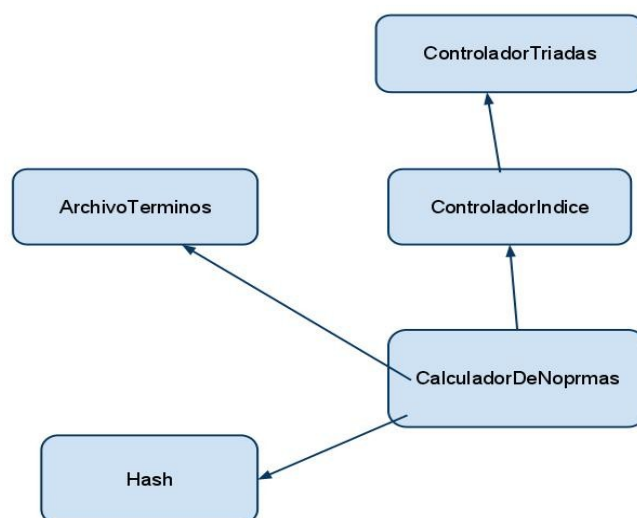
Una vez provisto un acceso eficiente a la información en disco a medida que se necesita, también fue necesario buscar la manera de que los cálculos intermedios de nuestras operaciones, muy valiosas en tiempos de accesos a disco, no se pierdan y que tengamos que recalcularlas cada vez. Recalcular, por ejemplo, el peso global de un término implica acceder al índice del árbol, levantar todas las tríadas del término, y calcular la cantidad de documentos que contienen dicho término, aparte del hecho de que los pesos globales se utilizan permanentemente, tanto para la generación del archivo de normas de documentos, como para los cálculos de similitudes, que se supone que no van a ser pocas. Por estas razones se decidió indexar los pesos globales, de manera que cada vez que se solicita generar un índice de normas de documentos, se genera antes el índice de pesos globales de términos.



En este gráfico se dibujó al módulo y los enlaces a la información con la que hace los cálculos. Los índices de pesos globales de términos y de normas de documentos, dependen exclusivamente del módulo, mientras que las estructuras dibujadas a la izquierda son las que nuestro módulo consulta.

Para solucionar el hecho de que el hash guarda números enteros, y los pesos globales de términos y normas de documentos son números reales, se corre la coma dos lugares al guardar y luego dos en sentido inverso al leer del índice, por lo tanto trabajamos con normas de documentos y pesos globales con dos decimales de precisión.

Clases del módulo



FrontCoding (Compresión del léxico)

El Front Coding es una tecnica utilizada para aprovechar informacion previa como base para nueva informacion que uno quiera ir almacenando.

Particularmente en este caso nos referimos a las palabras almacenadas en las hojas del arbol. Un ejemplo podria ser, almacenar dos claves tales como Sabato y Sabatella.

Haciendo un Front Coding estas quedarán como 0,6,Sabato y 5,4,ella; como se puede observar la segunda palabra se forma utilizando los primeros 5 caracteres de la anterior mas los 4 caracteres restantes que esta posee.

CaseFolding (Normalización de términos)

Para normalizar los términos, es decir, dejarlos de modo tal que para todo tipo de consulta y manipulación de los mismos su formato esté igual, se utilizan métodos dentro de la clase de servicio ServiceClass.

Luego de esta normalización, los términos quedan:

- en letra minúscula
- sin acentos
- canónicamente normalizados

Hash Expansible en disco

PARTICULARIDADES DE IMPLEMENTACIÓN

TABLA DE DIRECCIONES

Nuestro Hash Extensible en Disco se compone de una TABLA de direcciones, que está implementada sobre un archivo de registros fijos en disco. La TABLA se compone de BLOQUES.

Al recibirse una clave o palabra para almacenar, ésta se convierte a un valor numérico por medio de la FUNCIÓN DE HASH. Luego, a ese valor numérico se le aplica el operador de resto, mod, o '%' en C++, que nos devuelve el número de BLOQUE al que debe dirigirse el ELEMENTO que se inserta.

BLOQUE (clase DatoTablaHash*)

El bloque es la unidad de almacenamiento de la tabla de hash. Nuestro bloque está compuesto por solamente dos campos: OFFSETCUBETA y CANTIDADDEELEMENTOS. El primer valor, el más importante, guarda el offset en el archivo de datos a la CUBETA que contiene los datos del BLOQUE en cuestión.

ARCHIVO DE DATOS

Los datos se guardan efectivamente en un archivo de registros fijos. A cada registro se lo denomina CUBETA.

CUBETA (clase DatoCubetaHash*)

Cada CUBETA tiene escrita en su inicio metadata útil para su administración. Esta metadata consta de un número que indica los bytes libres para escribir datos en la cubeta (unsigned int bytesLibres), luego hay otro entero que indica el OFFSET A CUBETA DE EXTENSIÓN (uint32_t offsetProximaCubeta) y otro entero que indica la CANTIDAD DE ELEMENTOS (unsigned int cantElementos). La Clase provee servicios de hidratación, serialización y para la inserción y eliminación de ELEMENTOS.

EXTENSIÓN DE CUBETAS

Cuando nos quedamos sin espacio en una cubeta y se da el caso de que la función de Hash nos tira muchos elementos en la misma cubeta, y el crecimiento de la tabla no puede arreglar con practicidad el problema, se recurrió a extender la cubeta, seteando el campo OFFSET A CUBETA DE EXTENSION con un offset igual al de la nueva cubeta que se crea al final del archivo de cubetas para contener el elemento que no cupo en la cubeta original. Se forma así una SUCESIÓN DE CUBETAS ENCADENADAS para un único BLOQUE o DIRECCIÓN de la TABLA.

ELEMENTO (clase ElementoHash)

Esta clase es el dato unitario que se le ingresa y se le solicita al índice. Esta clase provee servicios de hidratación y serialización. En disco se escriben con metadata, que consta de un entero para indicar el TAMAÑO del elemento en disco (unsigned int tamañoEnDisco) y de otro entero que indica el OFFSET DE LIBRO, que contiene el offset al libro al que corresponde la palabra. Luego una cantidad de caracteres igual a TAMAÑO - 8 (8 = 2x4 bytes de metadata) para almacenar la palabra.

HASH EXTENSIBLE

Formatos de METADATA de estructuras en disco: <(tipo de dato) campo: x> ocupa x bytes.

DatoTablaHash:

<(uint32_t)offsetCubeta: 4><(unsigned int)cantidadElementos: 4>

DatoCubetaHash:

<(unsigned int)bytesLibres: 4><(uint32_t)offsetProxCubeta: 4><(unsigned int)cantElementos: 4><...

ElementoHash:

<(unsigned int)tamanoBytes:4><(uint32_t)offsetALibro: 4><('cadena')palabra: X>

Se puede interpretar el contenido de los archivos de BLOQUES (archivoTabla) y de CUBETAS (archivoCubetas) viendo la información con este formato. Se verifica que en efecto las palabras se guardan en cubetas y que son direccionadas por sus respectivos bloques en la tabla, y eventualmente que las cubetas se enlazan para extenderse.

EJEMPLOS

CUBETAS...

Así, en disco, en la posición de inicio de una cubeta (offset) en el archivo de cubetas, la siguiente información...

(1000)(0)(1)(12)(5900>('hola'))....

...significa que la cubeta actual tiene 1000 bytes libres para ingresar elementos, su offset de cubeta de extensión es CERO, lo cual nos INDICA QUE NO TIENE CUBETA DE EXTENSIÓN (la cubeta con offset cero es inicializada desde la creación del hash y nunca se elimina, nunca va a ser extensión de otra), y que esta cubeta tiene 1 (UN) elemento ingresado. Luego el entero (siempre de a 4 bytes) el 12 ya es parte de la metadata del elemento, y significa que este elemento ocupa 12 bytes en disco en total, 5900 es el offset al libro al que corresponde la palabra del elemento, y la palabra es 'hola'.

BLOQUES

En el archivo de bloques, suponiendo una tabla con dos entradas (por ejemplo, hash recién creado y vacío) al que se ingresó un elemento...

(0)(1)(1024)(0)

...indica que la cubeta del bloque '0' es 0, y tiene 1 elemento, que el offset de la cubeta del bloque '1' es 1024 (en un archivo de constantes se define TAMANIOCUBETA = 1024) y que no tiene ningún elemento. Siempre se toman los 'bloques' de a 8 bytes, y son 4 por cada entero.

Se recomienda para poder ver el contenido de los archivos usar el programa Ghex2.

Ver anexo: Diagrama de Clases de Hash Extensible.

Estructuras de Parseo

PARTICULARIDADES DE IMPLEMENTACIÓN

Se creó una clase padre ParserDeAtributo, que provee la interfaz para parsear un libro al indexador. De esta clase heredan ParserDePalabras, ParserDeAutor, ParserDeTitulo y ParserDeEditorial, que cumplen el propósito de parsear su correspondiente atributo al libro, a pedido del indexador, que responde a los comandos de consola del usuario.

Cada parser procesa el libro que le llega por parámetro en el método 'parsear', heredado de ParserDeAtributo y carga en una estructura de la clase Libro solamente el atributo que parseo. Este objeto Libro se usa como ODT (Object Data Transfer) y no tiene otro fin. Luego desde la aplicación se extrae el dato necesario del Libro y se genera el índice correspondiente.

FORMATO DE ARCHIVO DE STOPWORDS

El archivo que contiene las stopwords que debe ignorar el parser de palabras tiene el siguiente formato por cada línea:

`<stopword>\n`

Es decir, debe haber un fin de línea tras cada stopword, sin espacios en blanco, y una stopword por línea.

Casos de Prueba

1. Instrucción – Tomar Texto

- Se ingresa el comando *./TPDatos -i doc/prueba.txt*
- Se obtiene la instruccion ingresada como parametro
- Se creo el archivo de registros variables.
- Si el archivo no existe, se crea un archivo nuevo.
- Se cargan los libros.
- Se toma el archivo correspondiente.
- Se cierra el archivo de control.

Se puede verificar que en el archivo de control, hay un nuevo ingreso del tipo “0 | 0 | - | - | - | -” por ejemplo.

2. Instrucción – Procesar Editorial

- Se ingresa el comando *./TPDatos -e*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se crea el archivo del arbol
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- Se crea una copia del dato de control libro.
- Se comienzan a recuperar los libros y se los indexa según el tipo de indice ingresado, en este caso con la letra E.
- Se registran las nuevas indexaciones en el archivo de control.
- Se cierra el archivo de control.
- Se elimina el Arbol para liberar recursos.

Se puede verificar que en el archivo de control hay un nuevo ingreso del tipo “0 | 0 | E | - | - | - | -” . Se actualiza tambien el archivo correspondiente a este tipo de indice.

3. Instrucción – Procesar Autor

- Se ingresa el comando *./TPDatos -a*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se crea el archivo del arbol
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- Se crea una copia del dato de control libro.
- Se comienzan a recuperar los libros y se los indexa según el tipo de indice ingresado, en este caso con la letra A.
- Se registran las nuevas indexaciones en el archivo de control.
- Se cierra el archivo de control.
- Se elimina el Arbol para liberar recursos.

Se puede verificar que en el archivo de control, hay un nuevo ingreso del tipo “0 | 0 | A | - | - | - |” . Se actualiza tambien el archivo correspondiente a este tipo de indice.

4. Instrucción – Procesar Titulo

- Se ingresa el comando *./TPDatos -t*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- Se registran las nuevas indexaciones en el archivo de control.
- Se cierra el archivo de control.
- Se elimina el Hash

Se puede verificar que en el archivo de control hay ingresos sdel tipo “0 | 0 | T | - | - | - |” . Se actualiza tambien el archivo correspondiente a este tipo de indice.

5. Instrucción – Procesar Palabras

- Se ingresa el comando *./TPDatos -p*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- Se registran las nuevas indexaciones en el archivo de control.
- Se cierra el archivo de control.
- Se elimina el Hash


Se puede verificar que en el archivo de control, hay un nuevo ingreso del tipo “0 | 0 | P | - | - | - |” . Se actualiza tambien el archivo correspondiente a este tipo de indice.

6. Instrucción – Listar Archivos Tomados

- Se ingresa el comando *./TPDatos -l*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- A medida que se procesan las palabras se lee el archivo de stopwords para poder excluirlas.
- Se muestra por pantalla el siguiente texto por cada libro ingresado en la biblioteca:

TITULO: OTRO MAS TRES

AUTOR: Sagan

EDITORIAL: 

CANTIDAD PALABRAS REGISTRADAS: 7695

7. Instrucción - Obtener Archivo

- Se ingresa el comando *./TPDatos -o ID_Archivo*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se comienza a leer el archivo de control para obtener las referencias a los libros
- Se muestra por pantalla el archivo buscado, por ejemplo:

#TITULO_LIBRO PIRULO CON ACENTOS ◆

#AUTOR_LIBRO Asimoov

#EDITORIAL_LIBRO Zetha

Cap◆tulo Primero

LA CIENCIA DE LA DEDUCCI◆N

Sherlock Holmes cogi◆ la botella del ◆ngulo de la repisa de la chimenea,nar. Eso que a usted le resulta sorprendente, lo es tan s◆lo porque no sigue el curso de mis pensamientos, ni observa los hechos peque◆os de los que se pueden hacer deducciones importantes. Por ejemplo, empec◆ afirmando que su hermano era descuidado. Si se fija en la parte inferior de la tapa del reloj, observar◆ que no s◆lo tiene dos abolladuras, si no que muestra, tambi◆n, cortes y marcas por todas partes, debido a la costumbre de guardar en el
FIN

- Se cierra el archivo de control.

8. Instrucción – Quitar Archivo

- Se ingresa el comando *./TPDatos -q ID_Archivo*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.
- Se abre el archivo de control.
- Se cargan los libros.
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- Se recupera la indexacion de cada uno de los mismos y se elimina.
- Se actualiza el archivo de control.
- Se elimina el archivo.
- Se cierra el archivo de control.

Se puede verificar que en el archivo de control hay ingresos del tipo “0 | 100 | - | - | - | -” .

9. Instrucción – Ver Estructura

- Se ingresa el comando *./TPDatos -v -a resultado*
- Se obtiene la instruccion ingresada como parametro
- Se abre el archivo de registros variables.

- Se abre el archivo de control.
- Se cargan los libros.
- Se comienza a leer el archivo de control para obtener las referencias a los libros guardados.
- Se abre el archivo del arbol.
- Se muestra la estructura.

Se puede verificar que en el directorio de reportes se encuentran los archivos correspondientes a los indices y tablas realizados, los cuales reflejan las estructuras utilizadas.

Por ejemplo:

- Archivo de Estructura de Arbol - Resultado_Autor.txt

```
[B+Tree controlDataSize=[16] blockSize=[512] nodeCounter=[1] freeNodeCounter=[0] ]
|----- [node id=[0] level=[0] nodeSize=[83] controlDataSize=[16] nodeDataSize=[67] ]
| |----- [elemento KEY= [(0)(18)arthur conan doyle] LIBROS= [(530280)] sizeElemento=[30]
cantidadLibros=[1] ]
| |----- [elemento KEY= [(0)(8)cortazar] LIBROS= [(265145)] sizeElemento=[20] cantidadLibros=[1] ]
| |----- [elemento KEY= [(0)(5)sagan] LIBROS= [(0)] sizeElemento=[17] cantidadLibros=[1] ]
```

- Archivos de Estructuras de Hash

resultado_Titulo.txt_indice.txt

INDICE HASH

Nombre de archivo de tabla de hash: Indices/IndiceTitulo.dat

Nombre de archivo de cubetas de hash (datos): Indices/IndiceTitulo_Cubetas.dat

Cantidad de entradas de la tabla (bloques direccionables): 2

Cantidad de cubetas (unidades de almacenamiento de datos) efectivas: 2

resultado_Titulo.txt_libres.txt

REGISTRO DE ESPACIOS LIBRES

Se muestran las cubetas que no tienen Elementos (claves) guardados, aunque están siempre inicializadas y prestas a recibirlos

Bloque que contiene la cubeta libre: 0

Número de cubeta sucesiva del bloque: 1

Offset de la cubeta en el archivo: 0

Bloque que contiene la cubeta libre: 1

Número de cubeta sucesiva del bloque: 2

Offset de la cubeta en el archivo: 0

resultado_Titulo.txt_datos.txt

CUBETAS DE HASH

Cubeta del Bloque Número 0

Número de cubeta del mismo bloque: 1

Cantidad de Elementos (Claves) almacenados en la cubeta: 0

Bytes libres en la cubeta: 1012

Offset en disco de la cubeta Actual: 0 Offset de cubeta continuación: 0

Cubeta del Bloque Número 1
Número de cubeta del mismo bloque: 1
Cantidad de Elementos (Claves) almacenados en la cubeta: 1
Bytes libres en la cubeta: 991
Offset en disco de la cubeta Actual: 1024 Offset de cubeta continuación: 0

Cubeta del Bloque Número 1
Número de cubeta del mismo bloque: 2
Cantidad de Elementos (Claves) almacenados en la cubeta: 0
Bytes libres en la cubeta: 1012
Offset en disco de la cubeta Actual: 0 Offset de cubeta continuación: 0

resultado_Titulo.txt_tabla.txt TABLA DE HASH

Número de Bloque: 0
*** Cantidad de Elementos del bloque: 0
*** Offset a la Cubeta de Datos Inicial del bloque: 0

Número de Bloque: 1
*** Cantidad de Elementos del bloque: 1
*** Offset a la Cubeta de Datos Inicial del bloque: 1024

10. Instrucciones correspondientes a la segunda entrega

- Se deben dar permisos de ejecución al archivo test.sh mediante `chmod a+x test.s`
- Se debe correr test automatizados ejecutando el comando `./test.sh`

Para cualquier duda se puede consultar el archivo `readme.txt`

Reporte de BUGS

Como el calculo de normas no resulta del todo optimo en lo que respecta a la utilizacion de recursos, se decidio, a fines de mostrar el funcionamiento del trabajo practico, generar pruebas con libros menores a 500k. De lo contrario, el tiempo de calculo puede aumentar notoriamente.